This PDF Created by
JPG To PDF Converter for Mac
(Unregistered Version)

SPRING FRAMEWORK

Spring is grate framework for development of Enterprise grade applications. Spring is a light-weight framework for the development of enterprise-ready applications. Spring can be used to configure declarative transaction management, remote access to your logic using RMI or web services, mailing facilities and various options in persisting your data to a database. Spring framework can be used in modular fashion, it allows to use in parts and leave the other components which is not required by the application.

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

Spring Architecture

Spring is well-organized architecture consisting of seven modules. Modules in the Spring framework are:

1. **Spring AOP**
   One of the key components of Spring is the *AOP framework*. AOP is used in Spring:
   - 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 Spring's transaction abstraction.
   - To allow users to implement custom aspects, complementing their use of OOP with AOP
2. **Spring ORM**
   The *ORM* package is related to the database access. It provides integration layers for popular object-relational mapping APIs, including JDO, Hibernate and iBatis.
3. **Spring Web**
   The Spring Web module is part of Spring's web application development stack, which includes Spring MVC.
4. **Spring DAO**
   The DAO (Data Access Object) support in Spring is primarily for standardizing the data access work using the technologies like JDBC, Hibernate or JDO.
5. **Spring Context**
   This package builds on the beans package to add support for message sources and for the Observer design pattern, and the ability for application objects to obtain resources using a consistent API.

JPG to PDF Converter For Mac - Unregis

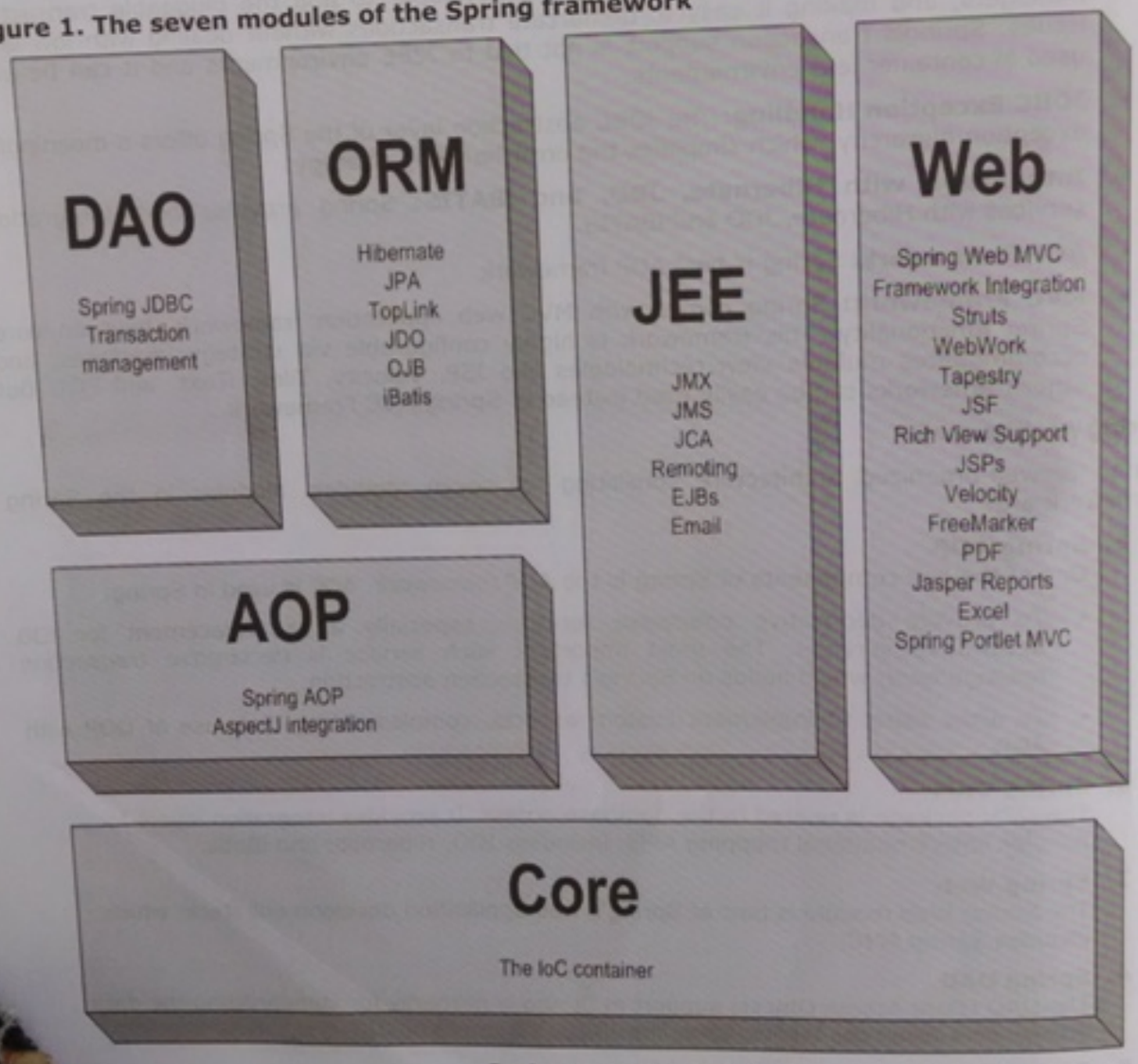
6. **Spring Web MVC**
   This is the Module which provides the MVC implementations for the web applications.

7. **Spring Core**
   The *Core* package is the most import component of the Spring Framework. This component provides the Dependency Injection features. The BeanFactory provides a factory pattern which separates the dependencies like initialization, creation and access of the objects from your actual program logic.

The following diagram represents the Spring Framework Architecture

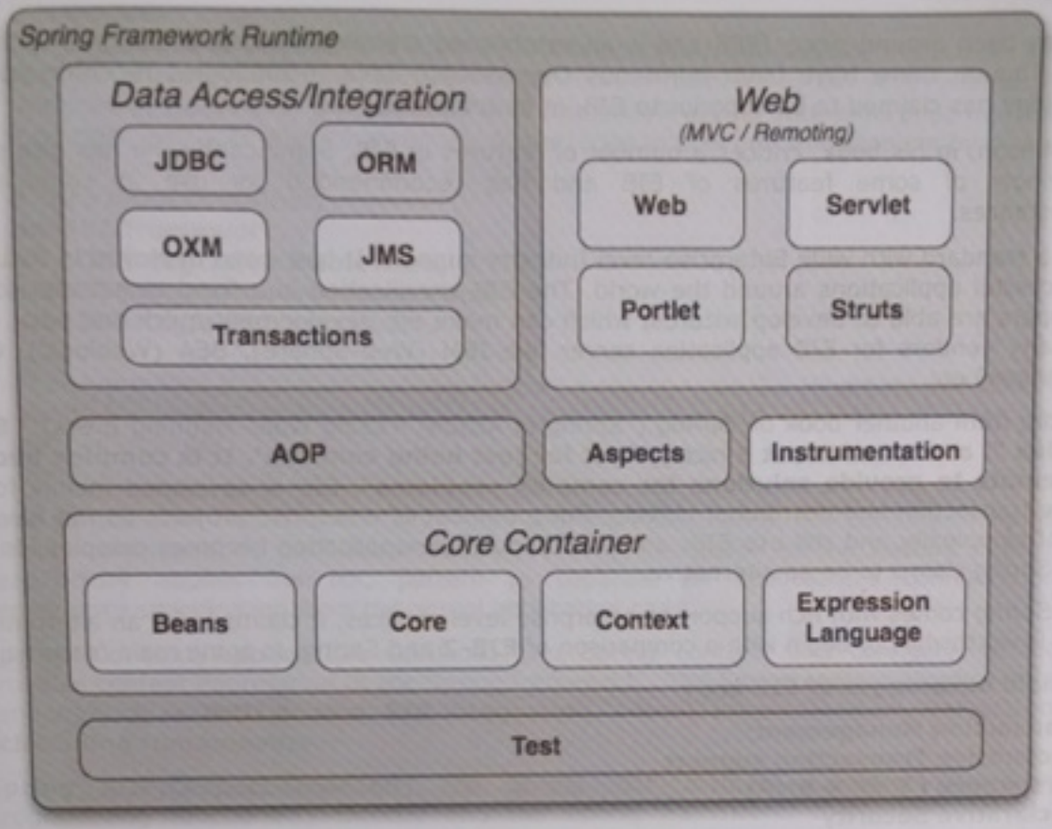
Spring1.x Framework Architecture

Figure 1. The seven modules of the Spring framework



Spring 2. X Architecture

JPG to PDF Converter For Mac - Unregis



Spring 3.x Architecture

Spring is an open-source application framework, introduced and developed in 2004. The main ideas were suggested by an experienced J2EE architect, **Rod Johnson**.

He had earlier, written a book titled '**J2EE Develoment without using EJB**' and had introduced the concept of Light-weight container. Primarily, his argument is that while EJB has its merits, it is not always necessary and suitable in all applications.

Just as Hibernate attacks CMP as primitive ORM technology, Spring attacks EJB as unduly complicated and not susceptible to unit-testing. Instead of EJB, Spring suggests that we make use of ordinary Java beans, with some slight modifications, to get all the supposed advantages of EJB environment. Thus, Spring is posed as an alternative to EJB essentially. However, as a concession to the existing EJB investments, Spring is designed to operate with EJB if required.

Much of the philosophy and approach of Spring framework, however, predates , the latest EJB vesrion (ie) EJB-3, about to arrive shortly. It is said that EJB-3 has absorbed a number of new ideas suggested by Spring and some more, to answer the criticisms. There is a debate going on in the Java community about Spring and EJB-3. Spring is not a Persistence technolgy but a framework which allows plug in of other such technologies. But **EJB-3** is primarily focussed on Persistence Technology and has now incorporated Hibernate, the best ORM todate.Talks are going on to incorpotrate another equally nice ORM Technology known as JDO, which provides for Object Database also. Moreoveer, EJB-3 's Attribute-Orientaed Meta tags, help in vastly reducing the size of XML lines. Some have complained that Spring is still too dependent on XML files. In this tutorial, any reference to EJB, is only to EJB-2.My aim in presenting this tutorial is not to advocate the choice of either Spring or EJB-3, a topic still being devbated by experts but to share my perception with our readers

In spring, we can make use of plain Java Beans to achieve things that were previously possible with EJB only. The main aim of Spring is to simplify the J2EE development and testing.

JPG to PDF Converter For Mac - Unregis

SPRING

EJB has been around since 1988 and is an established standard and specification in Enterprise world. Though there have been numerous Open-source Java technologies recently, no other technology has claimed to be superior to EJB, in its total features.

Rod Johnson, in his book, critices a number of features in EJB. Significantly, he has also spoken approvingly of some features of EJB and has recommended its use in some special circumstances.

EJB is a standard with wide Enterprise-level Industry support. It has been deployed in thousands of successful applications around the world. The EJB specification is a fixed target and so tool-developers are able to develop wizards, which can make ejb development quick and easy. There are many vendors for EJB application server like IBM (Web-Sphere), BEA (Weblogic), Oracle (JDeveloper) etc.

To quote from another book on Spring ("Spring in Action" - Craig Walls-Manning press) " **EJB is complex ", as the author put it nicely, "not for just being complex'. It is complex because it attempts to provide solutions for complex problems".** EJB is developed mainly for the remote transaction and distributed objects. But a number of enterprise projects do not have this level of complexity and still use EJBs and even the simple application becomes complex. In such cases Spring claims to be an alternative.

Since Spring comes with rich support to enterprise level services, it claims to be an alternative to EJB It is worthwhile to begin with a comparison of **EJB-2** and Spring, in some main features.

The main advantages of EJB are :

a) Transaction Management
b) Declarative Transaction support
c) Persistence (CMP & BMP)
d) Declarative Security
e) Distributed Computing (Container managed RPC)

Spring does not attempt to do everything by itself but supports the best of breed technologies for each of these requirements.

For example, instead of **CMP & BMP**, it supports several persistence technologies like **JDO, Hibernate and OJB**. These ORM toools are far more capable than the implementation in CMP.To simplify JDBC coding, there are tools like **iBatis** and Spring supports iBatis also.

Spring makes use of **Acegi**, an open-source Security framework and provides declarative security through spring configuration file or class **metadata** while in EJB declarative security is configured through deployment descriptor.

Spring provides proxying for **RMI (special remoting technologies like Burlap) JAX-RPC & web-service** while EJB provides container-managed remote method calls.

Spring can offer declarative transaction like EJB. But spring provides declarative rollback behavior also, for methods and exceptions

Thus, while EJB is monolithic and attempts to do many things, some tasks fairly well and some others not so well, Spring uses ordinary Java beans only and through special techniques provides many of the functionalities of EJB, by integrating with a number of open-source technologies.

Thereby, it gives the following advantages over **EJB2.**

a) testing is easier. We do not need to start the EJB container , for testing.

b) As Spring is based on the standard JavaBeans naming convention, programmers find it easy to work with.

c) It makes use of AOP(Aspect-Oriented Programming) which is a very recent and useful paradigm., in addition to classic OOP and preserves the purity of OOP.

JPG to PDF Converter For Mac - Unregis

d) It is flexible.
Spring's goal is to be an entire Application Framework. Other popular framewoks like **Struts, Tapestry, JSF** etc., are very good web tier frameworks but when we use these framework, we have to provide additional framework to deal with enterprise tier that integrates well with these framework. Spring tries to alleviate this problem by providing a comprehensive framework, which includes
a core bean container,
an MVC framework,
an AOP integration framework,
a JDBC integration framework and
an EJB integration framework.

It also provides integration modules for O/R mapping tools like Hibernate and JDO. Thus spring framework can be thought of as a layered architecture consisting of seven well defined modules.
The function of each component is as follows:

1. **Core Container**: The core container provides the fundamental functionality of Spring. It's primary component is the **'BeanFactory'**, an implementation of the Factory pattern. The BeanFactory applies the IOC pattern to separate an application's configuration and dependency specification from the actual application code.
2. **Spring Context/Application Context:** The Spring context is a configuration file that provides context information to the Spring framework . The Spring context supplies enterprise services such as **JNDI access, EJB integration, e-mail, internalization, validation, and scheduling functionality.**
3. **Spring AOP:(Aspect-Oriented)** The Spring AOP module integrates aspect-oriented programming functionality directly into the Spring framework, through its configuration management feature. As a result we 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 we can incorporate **declarative transaction management** into our applications without relying on EJB components. The Spring AOP module also introduces **metadata** programming to Spring. Using this we can add **annotation** to the source code that instructs Spring on where and how to apply aspects.
4. **Spring DAO:** The Spring's **JDBC and DAO** abstraction layer offers a meaningful exception hierarchy for managing the databaase connection, exception handling and error messages thrown by different database vendors. The exception hierarchy simplifies error handling and greatly reduces the amount of code that we need to write, such as opening and closing connections. This module also provide transaction management services for objects in a spring application.
5. **Spring ORM:** The Spring framework can be integrated to several ORM frameworks to provide Object Relational tool, including **JDO, Hibernate, OJB and iBatis SQL Maps.**
6. **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, JSF and webworks.** The Web module also eases the tasks of handling multipart requests and binding request parameters to domain objects.
7. **Spring MVC Framework:** The 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 and the generation of PDF and Excel Files.** I would venture to suggest that Spring will win sure acceptance among j2ee developers, very soon because of its ready-made adapters for various hot web-tier and presentation technologies.!

JPG to PDF Converter For Mac - Unregis

SPRING

For example, there is a great variety of technologies in the web-tier like MVC PATTERN, STRUTS, JSF, WEB-WORK, JSP, TAPESTRY,FREEMARKER etc. Developers are now puzzled and confused about the relative merits and demerits of all these. Once they choose a technology and start implementing and later want to change over to another technology, it is very difficult. But, as Spring offers modules for aall the above technologies, it is most often simply changing the configuraion file. With this approach, it is even possible for a development team to try and test a given task in all the above forms and see the effect and performance before deciding the choice. Spring offers its own version of MVC architecture. It also offers adapters for Struts.

In the MVC adapter, it offers the following View choices.

JSP is default view template. **'InternalResouceViewResolver'** can be used for this purpose. Spring can be integrated with other template solutions like **Velocity, FreeMarker, tiles** etc., Also Spring can be used to produce dynamic binary **Excel** spreadsheet, **PDF** documents etc.,

To configure the velocity engine **'VelocityConfigurer'** bean is declared in spring configuration. The view resolver is configured by **'VelocityViewResolver'** bean.

To configure the **FreeMarker** engine **'FreeMarkerConfigurer'** bean is declared in spring configuration. The view resolver is configured by **'FreeMarkerViewResolver'** bean.

'TilesConfigurer' can used to used to load Tiles configuration file for rendering Tiles view. **'AbstactPdfView'** **'AbstractExcelView'** is used to generate Excel SpreadSheet as views. **'buildPdfDocument()'** is used to create PDF decument. supports the creation of PDF as views. Similarly we have **'buildExcelDocument()'** to create the excel document.

For delegation purpose, Spring provides **'DelegatingRequestProcessor'** and to use the tiles **'DelegatingTilesRequestProcessor'** is used. **'SpringTapestryEngine'** is used for integrating Tapestry to Spring. **'FacesSpringVariableResolver'** is used to resolve spring-managed beans in JSF.

Next we shall see the main concepts of Spring, **Inversion of Control (IoC) and Aspect Oriented Programming**. Spring is based on dependency injection type of IoC . We don't directly connect our components and services together **in code** but describe which services are needed by which components in a **configuration file**. A container is responsible for hooking it up. This concept is similar to **'Declarative Management'**. IOC is a broad concept. the two main types are

1. **Dependency Lookup:**

   The container provides callbacks to components and a lookup context. The managed objects are responsible for their other lookups. This is the **EJB Approach.** The Inversion of Control is limited to the Container involved callback methods that the code can use to obtain resources. Here we need to use JNDI to look up other EJBs and resources. Because of this reason EJB is not branded as 'IOC framework'**.** There are some problems in this implementation. The class **needs a application server environment** as it is dependent on JNDI and it is hard to test as we need to provide a dummy JNDI contest for testing purpose.

2. **Dependency Injection:**

   In this application objects is not responsible for looking up resources they depend on. Instead IoC container configures the object externalizing resource lookup from application code into the container. That is, **dependencies are injected into objects.** Thus lookups are completely removed from application objects and it can be used **outside the container also.** In this method, the objects can be populated via **Setter Injection** (Java-Beans properties) or **Constructor Injection** (constructor arguments). Each method has its own advantage and disadvantage.

JPG to PDF Converter For Mac - Unregis

Normally in all the java beans, we will use setter and getter method to set and get the value of property as follows

```
public class namebean
{
   String     name;

   public void setName(String a)
   {
     name = a;
   }
   public String getName()
   {
     return name;
   }
}
```

We will create an instance of the bean 'namebean' (say bean1) and set property as **bean1.setName("tom");** Here in setter injection, we will set the property 'name' by using the <property> subelement of <bean> tag in spring configuration file as showm below,

```
<bean id="bean1"   class="namebean">

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

</bean>
```

The subelement <value> sets the 'name' property by calling the set method as **setName("tom");** This process is called **setter injection.**

For constructor injection, we use constructor with parameters as shown below,

```
public class namebean
{
    String name;
    public namebean(String a)
    {
      name = a;
    }
}
```


JPG to PDF Converter For Mac - Unregis

We will set the property 'name' while creatinf an instance of the bean 'namebean' as **namebean bean1 = new namebean("tom");**

Here we use the **<constructor-arg>** element to set the the property by constructor injection as

```
<bean id="bean1"  class="namebean">
  <constructor-arg>
    <value>My Bean Value</value>
  </constructor-arg>
</bean>
```

To set properties that reference other beans **<ref>,** subelement of <property> is used as shown below,

```
<bean id="bean1"  class="bean1impl">
  <property name="game">
    <ref bean="bean2"/>
  </property>
</bean>
<bean id="bean2"  class="bean2impl" />
```

Aspect-Oriented Programming

Aspect Oriented programming is a new

programming technique that promotes separation of concerns within the system. System are composed of several components each responsible for a specific piece of functionality. Whatever may be the core function of the program, the system service like logging, transaction management, security etc., must be included in the program. These system services are commonly refered to as **'cross-cutting concerns'** as they tend to cut across multiple components in a system. **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. In spring, aspects are **wired** into objects in the spring XML file in the same way as JavaBean. This process is known as **'Weaving'**.

The container is at the core of Spring Container. In manages the life cycle and configuration of application objects. We can configure how each of our beans should be created either to create a single instance of bean or produce a new instance every time and how they should be associated with each other. Spring should not, however, be confused with traditionally heavyweight EJB containers, which are often large. The Spring actually comes with two distinct containers: Bean Factories-defined by **"org.springframework. beans.factory.BeanFactory"** are the simplest containers, providing support for dependency injection. Application contexts - defined by **"org.springframework.context.Application Context"** provides application framework services.

BEAN FACTORY:

Bean factory is an implementation of the factory design pattern and its function is to create and dispense beans. As the bean factory knows about many objects within an application, it is able to create association between collaborating objects as they are instantiated. This removes the burden of configuration from the bean and the client.

There are several implementation of BeanFactory. The most useful one is **"org.springframework.beans.factory.xml. XmlBeanFactory"**. It loads its beans based on the definition contained in an XML file. To create

JPG to PDF Converter For Mac - Unregis

an **XmlBeanFactory**, pass a **InputStream** to the constructor. The resource will provide the XML to the factory.

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

This line tells the bean factory to read the bean definition from the XML file. The bean definition includes the description of beans and their properties. But the bean factory doesn't instantiate the bean yet. To retrieve a bean from a 'BeanFactory', the getBean() method is called. When getBean() method is called, factory will instantiate the bean and begin setting the bean's properties using dependency injection.

myBean bean1 = (myBean)factory.getBean("myBean");

APPLICATION CONTEXT:

While Bean Factory is used for simple applications, the Application Context is spring's more advanced container. Like 'BeanFactory' it can be used to load bean definitions, wire beans together and dispense beans upon request.

It also provide

1) a means for resolving text messages, including support for internationalization.

2) a generic way to load file resources.

3) events to beans that are registered as listeners.

Because of additional functionality, 'Application Context' is preferred over a BeanFactory. Only when the resource is scarce like mobile devices, 'BeanFactory' is used. The three commonly used implementation of 'Application Context' are

1. 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");
2. 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");
3. XmlWebApplicationContext : It loads context definition from an XML file contained within a web application.

Spring is lightweight in terms of both size and overhead. The entire Spring framework can be distributed in a single JAR file that weighs just over **1.7 MB.** And the processing overhead required by Spring is negligible. Also Spring is nonintrusive:(ie) objects in a Spring-enabled application typically have no dependencies on Spring-specific classes.

There are other lightweight containers like **HiveMind, Avalon, PicoContainer** etc., **Avalon** was one of the first IoC containers. Avalon mainly provides interface-dependent IoC. so, we much change our code in order to use a different container. This couples your code to a particular framework which is an undesirable feature.

PicoContainer is a minimal (very small size nearly 50k) lightweight container that provides IoC in the form of constructor and setter injection. By using PicoContainer we can only assemble components programmatically through PicoContainer's API. But it allows only one instance of any particular type to be present in the registry. Also PicoContainer is only a container. It does not offer various special features as spring like integration.

HiveMind is relatively new IoC container. Like PicoContainer, it focuses on wiring with support for both constructor and setter injections. It also allows us to define our configuration in an XML file. Like PicoContainer, HiveMind is only a container. It does not offer integration with other technology. Thus Spring makes it possible to configure and compose complex applications from simpler components. In Spring, application objects are composed declaratively, typically in an XML file. Spring also provides much infrastructure functionality

JPG to PDF Converter For Mac - Unregis

like transaction management, persistence framework integration, etc., leaving the development of application logic to us.

With this inroduction, we shall see a simple example in Spring in next article.

The **Spring Framework** comes in the form of ZIP file with the necessary jars, examples etc., The Spring framework can be downloaded from **http://www.springframework.org.** There will be two zip files one with dependencies and other without. The spring framework with dependencies is larger and includes all dependent libraries. Download **Spring framework 1.2 without dependency** and unzip on the hard disk as spring12

Inside the folder spring12 we can find 7 folders. The name and the contents of all the folders are given below,

1. **dist:** It contains various Spring distribution jar files.
2. **docs:** It contains general documentation and API javadocs.
3. **mock:** It contains mock JNDI contexts and a set of servlet API mock objects.
4. **samples:** It contains demo applications and skeletons.
5. **src:** It contains the Java source files for the framework.
6. **test:** It contains the Java source files for Spring's test suite.
7. **tiger:** It contains JDK1.5 examples and test suite.

Inside the "dist" directory, we can find all the jar files necessary for compiling and executing the program. The jar files and its contents are listed below:

1. **spring.jar :** It contains the entire Spring framework including everything in the other JAR files also.
2. **spring-core.jar** : It contains the core Spring container and its utilities.
3. **spring-beans.jar :** It contains the bean Spring container and JavaBeans support utilities.
4. **spring-aop.jar :** It contains Spring's AOP framework, source-level metadata support, AOP Alliance interfaces etc.,
5. **spring-context.jar :** It contains application context, validation framework, JNDI, templating support and scheduling.
6. **spring-dao.jar :** It contains DAO support and transaction infrastructure.
7. **spring-jdbc.jar :** It contains the JDBC support.
8. **spring-support.jar :** It contains JMX support, JCA support, scheduling support, mail support and caching support.
9. **spring-web.jar :** It contains the web application context, multipart resolver, Struts support, JSF support and web utilities.
10. **spring-webmvc.jar :** It contains the framework servlets, web MVC framework, web controllers and web views.
11. **spring-remoting.jar :**It contains remoting support, EJB support and JMS support.
12. **spring-orm.jar :** It contains iBATIS SQL Maps support, Apache OJB support, TopLink support and JDO support.
13. **spring-hibernate.jar :** It contains Hibernate 2.1 support, Hibernate 3.x support.
14. **spring-mock.jar :** It contains JNDI mocks, Servlet API mocks and JUnit support.

JPG to PDF Converter For Mac - Unregis

Now we will run a greeter example in Spring.

f:\>md springdemo

f:\>cd springdemo

As the entire Spring Framework is included in spring.jar. We use that to run our examples. Copy **spring.jar** from spring12\dist folder to the working folder. Also copy **commons-logging.jar** from tomcat41\server\lib to the working directory.

f:\springdemo>set path=c:\windows\command;**g:\jdk1.5\bin**
(* Set path for jdk1.5 or jdk1.4.2 only.)
f:\springdemo>set classpath=f:**springdemo;**
f:\springdemo**spring.jar;**

f:\springdemo**commons-logging.jar**

For a typical Spring Application we need the following files

1. An **interface** that defines the functions.
2. An **Implementation** that contains properties, its setter and getter methods, functions etc.,
3. A XML file called **Spring configuration file.**
4. Client program that uses the function.

Edit

1. hello.java
2. helloimpl.java
3. hello.xml
4. helloclient.java

----------f:\springdemo\hello.java----------

```
public interface hello
{
  public String sayhello(String a);
}
```

----------f:\springdemo\helloimpl.java----------

```
public class helloimpl implements hello
{
    private String greeting;
    public helloimpl()
    {
    }
    public helloimpl(String a)
    {
    greeting=a;
    }
    public String sayhello(String s)
    {
        return greeting+s;
    }
    public void setGreeting(String a)
    {
```


JPG to PDF Converter For Mac - Unregis

```
        greeting=a;
    }
}
```

----------f:\springdemo\hello.xml----------

```
<?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="hello"
        class="helloimpl">
   <property name="greeting">
      <value>Good Morning!...</value>
   </property>
   </bean>
  </beans>
```

----------f:\springdemo\helloclient.java----------

```
import java.io.*;
import org.springframework.beans.factory.*;
import org.springframework.beans.factory.xml.*;
import org.springframework.core.io.*;
public class helloclient
{
   public static void main(String args[])  throws Exception
   {
      try
      {
         System.out.println("please Wait.");
         Resource     res = new  ClassPathResource("hello.xml");
         BeanFactory  factory = new  XmlBeanFactory(res);
         hello bean1 = (hello)factory.getBean("hello");
         String s = bean1.sayhello(args[0]);
         System.out.println(s);
      }
      catch(Exception e1)
            { System.out.println(""+e1); }
   }
}
```

To run:

```
f:\springdemo>javac hello.java
```

JPG to PDF Converter For Mac - Unregis

```
f:\springdemo>javac helloimpl.java
f:\springdemo>javac helloclient.java
f:\springdemo>java helloclient "sam"
```

We will get the output as follows:

```
please wait..
Aug 5, 2002 2:10:56 AM org.springframework.
beans.factory.xml.XmlBeanDefinitionReader
loadBeanDefinitions

INFO: Loading XML bean definitions from
class path resource [hello.xml]
Aug 5, 2002 2:10:57 AM org.springframework.
beans.factory.support.
AbstractBeanFactorygetBean

INFO: Creating shared instance of singleton
bean 'hello'
Good Morning!...sam
```

Thus we have run our first Spring program. Here helloimpl implements the hello interface. Although it is not necessary to hide the implementation behind an interface, it is recommended as a way to seperate implementation from interface. helloimpl has a single property greeting. This property can be set in two different ways: by property's setter method or by the constructor. The XML file hello.xml declares the instance of helloimpl.java in the spring container and configures its property greeting with the value 'Good Morning!...'

The root of the hello.xml file is the <beans> element, which is the root element of any Spring configuration file. The <bean> element is used to tell the Spring container about the class and how it should be configured. Here, the id attribute takes the interface name and the class attribute specifies the bean's fully qualified class name.

Within the <bean> element, the <property> element is used to set the property, in this case the greeting property. By using <property>, we're telling the Spring container to call setGreeting() while setting the property. The value of the greeting is defined within the <value> element. Here we have given **'Good Morning!...'** as the value.

The container instantiates the 'helloimpl' based on the XML definition as,

```
helloimpl hello = new helloimpl();
helloimpl.setGreeting("Good Morning!...");
```

Similarly, greeting property may be set through the single argument constructor of 'helloimpl' as,

```
<bean id="hello"  class="helloimpl">
<constructor-arg>
<value>Good Morning!...</value>
</constructor-arg>
</bean>
```

Now the container instantiates the 'helloimpl' based on the XML definition as,

```
helloimpl hello = new
helloimpl("Good Morning...");
```

In the client program, '**BeanFactory**' class is used which is the Spring container. Then the 'getBean()' method is called to get the reference to the 'hello'. With this reference, sayhello() method is called.

We can also use a frame client. The code is very much similar to console client except the GUI code.

JPG to PDF Converter For Mac - Unregis

----------f:\springdemo\helloframe.java----------

```
import java.io.*;
import java.awt.*;
import org.springframework.beans.factory.*;
import org.springframework.beans.factory.xml.*;
import org.springframework.core.io.*;
public class helloframe  extends Frame
{
TextField     text1;
TextArea      area1;
Label         label1;
Button        button1;
   public static void main(String args[])
   {
     helloframe  app = new helloframe();
     app.setSize(700,500);
     app.setVisible(true);
   }
   helloframe()
   {
     setLayout(new FlowLayout());
     setBackground(Color.green);
     label1=new Label("Type Name: ");
     text1=new TextField(25);
     area1=new TextArea(10,50);
     button1=new Button("Exit");
     button1.setBackground(Color.red);
     add(label1);
     add(text1);
     add(area1);
     add(button1);
   }
   public boolean action (Event e,Object c)
   {
     if(e.target==text1)
     {
       try
       {
```


JPG to PDF Converter For Mac - Unregis

```
        area1.append("Please Wait..\n");
        Resource res = new ClassPathResource("hello.xml");
        BeanFactory factory = new XmlBeanFactory(res);
        hello bean1 = (hello)factory.getBean("hello");
        String s = bean1.sayhello(text1.getText());
        area1.append(s);
      }
      catch(Exception e1)
            {area1.append(""+e1);}
    }
    if(e.target==button1)
    {
      System.exit(0);
    }
    return true;
  }
```

Then compile and run the frame client program. We will a textbox, a exit button and a text area. Type a name (say 'tom') in text area and press enter. **'Good Morning!... tom'** will appear in the text area

Next we shall see how to run this program as a servlet. Consider Tomcat-5 is installed in g drive.

First copy g:\spring12\spring.jar to g:\tomcat5\common\lib and start tomcat server.

Then set classpath as shown below and edit the servletclient.java.

f:\springdemo>set classpath=f:**springdemo;**

f:\springdemo**spring.jar;**

f:\springdemo**commons-logging.jar;**

g:\tomcat5\common\lib\servlet-api.jar

----------f:\springdemo\servletclient.java ----------

```
import java.io.*;
import org.springframework.beans.factory.*;
import org.springframework.beans.factory.xml.*;
import org.springframework.core.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class servletclient extends HttpServlet
{
  public void doPost(HttpServletRequest req,HttpServletResponse resp)
      throws ServletException,IOException

  {
    resp.setContentType("text/html");
    PrintWriter out =resp.getWriter();
```


JPG to PDF Converter For Mac - Unregis

```
String a = req.getParameter("text1");
try
{
  System.out.println("Please wait.");

  Resource res = new ClassPathResource("hello.xml");
  System.out.println("Resource ok");

  BeanFactory factory = new XmlBeanFactory(res);
  System.out.println("BeanFactory ok");

  hello bean1 = (hello)factory.getBean("hello");

  String s = bean1.sayhello(a);
  out.println(s);
 }
 catch(Exception e1)
      {System.out.println(""+e1);}
 }
}
```

----------f:\springdemo\servletclient.htm----------

```
<html>
<body>
<form    method=post
      action="http://localhost:8080/
              servlet/servletclient">
  <input type=text name="text1">
  <input type=submit>
</form>
</body>
</html>
```

Then compile the servlet and copy all the class files ie., hello.class, helloimpl.class, servletclient.class and the xml file hello.xml to g:\tomcat5\webapps\root\web-inf\classes. Copy html file servletclient.htm to g:\tomcat5\webapps\root. Add entry to web.xml file. Restart Tomcat server and open browser and type url as http://localhost:8080/servletclient.htm We will get a text box and a button. Type a name (say 'tom') and click the 'submit' button.

Good Morning!... tom will appear

In the next article, we shall see how to contact the database from Spring.

JPG to PDF Converter For Mac - Unregis

IOC Container

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.

In short, the 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.

In Spring, those 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; other than that, there is nothing special about a bean (it is in all other respects one of probably many objects in your application). These beans, and the dependencies between them, are reflected in the configuration metadata used by a container.

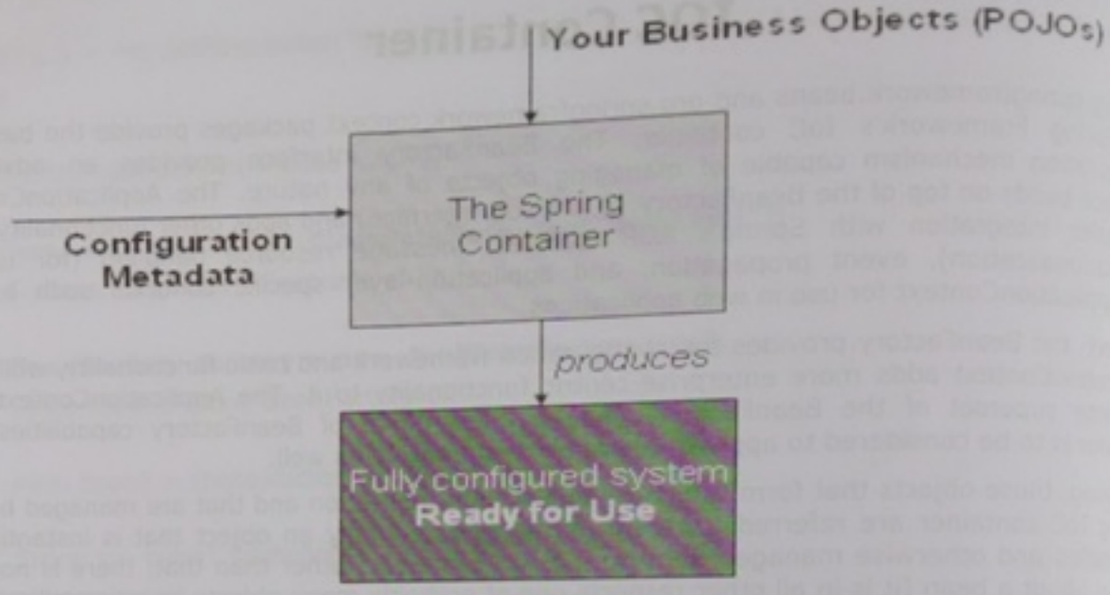
The org.springframework.beans.factory.BeanFactory is the actual representation of the Spring IoC container that is responsible for containing and otherwise managing the aforementioned beans. The BeanFactory interface is the central IoC container interface in Spring. Its responsibilities include instantiating or sourcing application objects, configuring such objects, and assembling the dependencies between these objects.

There are a number of implementations of the BeanFactory interface that come supplied straight out-of-the-box with Spring. The most commonly used BeanFactory implementation is the XmlBeanFactory class. This implementation allows you to express the objects that compose your application, and the doubtless rich 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 consumes some form of configuration metadata; this configuration metadata is nothing more than how you (as an application developer) inform the Spring container as to how to "instantiate, configure, and assemble [the objects in your application]". This configuration metadata is typically supplied in a simple and intuitive XML format. When using XML-based configuration metadata, you write bean definitions for those beans that you want the Spring IoC container to manage, and then let the container do its stuff.

XML-based metadata is by far the most commonly used form of configuration metadata. It is **not** however the only form of configuration metadata that is allowed. Spring configuration consists of at least one bean definition that the container must manage, but typically there will be more than one bean definition. When using XML-based configuration metadata, these beans are configured as <bean/> elements inside a top-level <beans/> element. These bean definitions correspond to the actual objects that make up your application.

JPG to PDF Converter For Mac - Unregis



The Spring IoC container

Example of the basic structure of XML-based configuration metadata.

```
<?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-2.5.xsd">

        <bean id = "..."  class = "...">
                <!-- collaborators and configuration for this bean go here -->
        </bean>

        <!-- more bean definitions go here -->
</beans>
```

Beans

A Spring IoC container manages one or more beans. These beans are created using the configuration metadata that has been supplied to the container (typically in the form of XML <bean/> definitions). Within the container itself, these bean definitions are represented as BeanDefinition objects, which contain (among other information) the following metadata:

- a package-qualified class name: typically this is the actual implementation class of the bean being defined.

JPG to PDF Converter For Mac - Unregis

- bean behavioral configuration elements, which state how the bean should behave in the container (scope, lifecycle callbacks, and so forth).
- references to other beans which are needed for the bean to do its work; these references are also called collaborators or dependencies.
- other configuration settings to set in the newly created object. An example would be the number of connections to use in a bean that manages a connection pool, or the size limit of the pool.

Naming beans - Every bean has one or more ids (also called identifiers, or names; these terms refer to the same thing). These ids must be unique within the container the bean is hosted in. A bean will almost always have only one id, but if a bean has more than one id, the extra ones can essentially be considered aliases. When using XML-based configuration metadata, you use the 'id' or 'name' attributes to specify the bean identifier(s). The 'id' attribute allows you to specify exactly one id, and as it is a real XML element ID attribute, the XML parser is able to do some extra validation when other elements reference the id; as such, it is the preferred way to specify a bean id. However, the XML specification does limit the characters which are legal in XML IDs. If you have a need to use one of these special XML characters, or want to introduce other aliases to the bean, you may also or instead specify one or more bean ids, separated by a comma (,), semicolon (;), or whitespace in the 'name' attribute.

Aliasing beans - In a bean definition itself, you may supply more than one name for the bean, by using a combination of up to one name specified via the id attribute, and any number of other names via the name attribute. All these names can be considered equivalent aliases to the same bean, and are useful for some situations, such as allowing each component used in an application to refer to a common dependency using a bean name that is specific to that component itself.

Consider the case where component A defines a DataSource bean called componentA-dataSource, in its XML fragment. Component B would however like to refer to the DataSource as componentB-dataSource in its XML fragment. And the main application, MyApp, defines its own XML fragment and assembles the final application context from all three fragments, and would like to refer to the DataSource as myApp-dataSource. This scenario can be easily handled by adding to the MyApp XML fragment the following standalone aliases:

```
<alias name="componentA-dataSource" alias="componentB-dataSource"/>
<alias name="componentA-dataSource" alias="myApp-dataSource" />
```

Instantiating beans - A bean definition essentially is a recipe for creating one or more objects. The container looks at the recipe for a named bean when asked, and uses the configuration metadata encapsulated by that bean definition to create (or acquire) an actual object.

- Using Constructor

```
<bean id="exampleBean" class="examples.ExampleBean"/>
<bean name="anotherExample" class="examples.ExampleBeanTwo"/>
```

- Using Static Factory method

```
<bean id="exampleBean" class="examples.ExampleBean" factory-method="createInstance"/>
```

- Using Instance Factory method

```
<!-- the factory bean, which contains a method called createInstance() -->
<bean id="serviceLocator" class=" DefaultServiceLocator" />

<!-- the bean to be created via the factory bean -->
```


JPG to PDF Converter For Mac - Unregis

```
<bean id="exampleBean"
        factory-bean="serviceLocator"
        factory-method="createInstance"/>
```

- Using Container

```
Resource res = new FileSystemResource("beans.xml");
BeanFactory factory = new XmlBeanFactory(res);
```

Dependencies

The basic principle behind **Dependency Injection** (DI) is that objects define their dependencies (that is to say the other objects they work with) only through constructor arguments, arguments to a factory method, or properties which are set on the object instance after it has been constructed or returned from a factory method. Then, it is the job of the container to actually inject those dependencies when it creates the bean.

This is fundamentally the inverse, hence the name **Inversion of Control** (IoC), of the bean itself being in control of instantiating or locating its dependencies on its own using direct construction of classes, or something like the Service Locator pattern. It becomes evident upon usage that code gets much cleaner when the DI principle is applied, and reaching a higher grade of decoupling is much easier when objects do not look up their dependencies. DI exists in two major variants, namely Constructor Injection and Setter Injection.

Constructor Injection

```
package x.y;
public class Foo {
    public Foo(Bar bar, Baz baz) {
        // ...
    }
}
```

```
<beans>
    <bean name="foo" class="x.y.Foo">
        <constructor-arg>
            <bean class="x.y.Bar"/>
        </constructor-arg>
        <constructor-arg>
            <bean class="x.y.Baz"/>
        </constructor-arg>
    </bean>
</beans>
```


JPG to PDF Converter For Mac - Unregis

```
package examples;
public class ExampleBean
{
    private int years;
    private String ultimateAnswer;

    public ExampleBean(int years, String ultimateAnswer) {
        this.years = years;
        this.ultimateAnswer = ultimateAnswer;
    }
}
```

Constructor Argument Type Matching

```
<bean id="exampleBean" class="examples.ExampleBean">
    <constructor-arg type="int" value="7500000"/>
    <constructor-arg type="java.lang.String" value="42"/>
</bean>
```

Constructor Argument Index

```
<bean id="exampleBean" class="examples.ExampleBean">
    <constructor-arg index="0" value="7500000"/>
    <constructor-arg index="1" value="42"/>
</bean>
```

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.

How Injection works?

- The BeanFactory is created and initialized with a configuration which describes all the beans. (Most Spring users use a BeanFactory or ApplicationContext implementation that supports XML format configuration files.)
- Each bean has dependencies expressed in the form of properties, constructor arguments, or arguments to the static-factory method when that is used instead of a normal constructor. These dependencies will be provided to the bean, when the bean is actually created.
- Each property or constructor argument is either an actual definition of the value to set, or a reference to another bean in the container.
- Each property or constructor argument which is a value must be able to be converted from whatever format it was specified in, to the actual type of that property or constructor

JPG to PDF Converter For Mac - Unregis

argument. By default Spring can convert a value supplied in string format to all built-in types, such as int, long, String, boolean, etc.

The Spring container validates the configuration of each bean as the container is created, including the validation those properties which are bean references are actually referring to valid beans. However, the bean properties themselves are not set until the bean is actually created. For those beans that are singleton-scoped and set to be pre-instantiated (such as singleton beans in an ApplicationContext), creation happens at the time that the container is created, but otherwise this is only when the bean is requested. When a bean actually has to be created, this will potentially cause a graph of other beans to be created, as its dependencies and its dependencies' dependencies (and so on) are created and assigned.

Inner beans

A <bean/> element inside the <property/> or <constructor-arg/> elements is used to define a so-called inner bean. An inner bean definition does not need to have any id or name defined, and it is best not to even specify any id or name value because the id or name value simply will be ignored by the container.

```
<bean id="outer" class="...">
    <!-- instead of using a reference to a target bean, simply define the target bean inline -->
    <property name="target">
        <bean class="com.example.Person"> <!-- this is the inner bean -->
            <property name="name" value="Fiona Apple"/>
            <property name="age" value="25"/>
        </bean>
    </property>
</bean>
```

Note that in the specific case of inner beans, the 'scope' flag and any 'id' or 'name' attribute are effectively ignored. Inner beans are always anonymous and they are always scoped as prototypes. Please also note that it is not possible to inject inner beans into collaborating beans other than the enclosing bean.

Compound property names

Compound or nested property names are perfectly legal when setting bean properties, as long as all components of the path except the final property name are not null.

```
<bean id="foo" class="foo.Bar">
    <property name="x.y.z" value="123" />
</bean>
```

The foo bean has a x property which has a y property, which has a z property, and that final z property is being set to the value 123. In order for this to work, the x property of foo, and the y property of x must **not** be null after the bean is constructed, or a NullPointerException will be thrown.

depends-on

For most situations, the fact that a bean is a dependency of another is expressed by the fact that one bean is set as a property of another. This is typically accomplished with the <ref/> element in XML-based configuration metadata. For the relatively infrequent situations where dependencies between beans are less direct (for example, when a static initializer in a class needs to be triggered, such as database driver registration), the 'depends-on' attribute may be used to

JPG to PDF Converter For Mac - Unregis

explicitly force one or more beans to be initialized before the bean using this element is initialized. Find below an example of using the 'depends-on' attribute to express a dependency on a single bean.

```
<bean id="beanOne" class="ExampleBean" depends-on="manager"/>
<bean id="manager" class="ManagerBean" />
```

If you need to express a dependency on multiple beans, you can supply a list of bean names as the value of the 'depends-on' attribute, with commas, whitespace and semicolons all valid delimiters, like so:

```
<bean id="beanOne" class="ExampleBean" depends-on="manager,accountDao">
    <property name="manager" ref="manager" />
</bean>

<bean id="manager" class="ManagerBean" />
<bean id="accountDao" class="x.y.jdbc.JdbcAccountDao" />
```

The 'depends-on' attribute and property is used not only to specify an initialization time dependency, but also to specify the corresponding destroy time dependency (in the case of singleton beans only). Dependent beans that are defined in the 'depends-on' attribute will be destroyed first prior to the relevant bean itself being destroyed. This thus allows you to control shutdown order too.

JPG to PDF Converter For Mac - Unregis

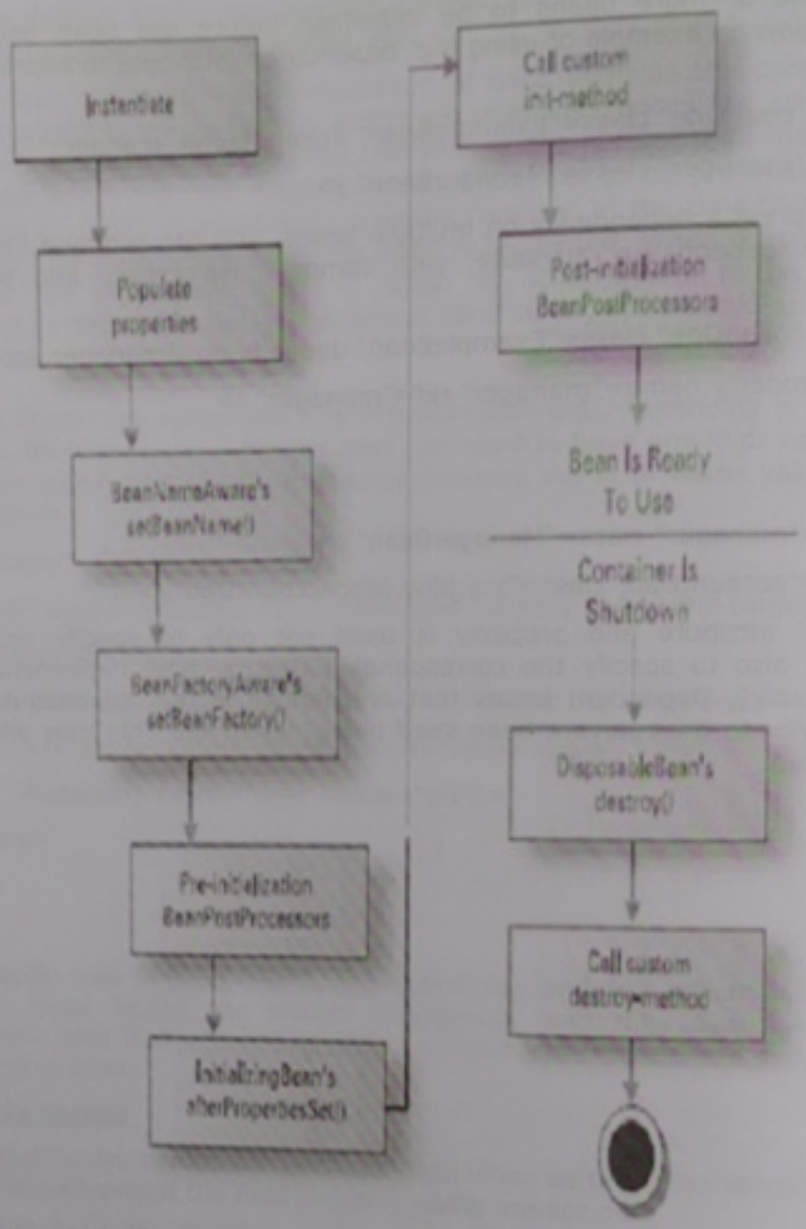


Figure 2.1 The life cycle of a bean within a Spring bean factory container

JPG to PDF Converter For Mac - Unregis

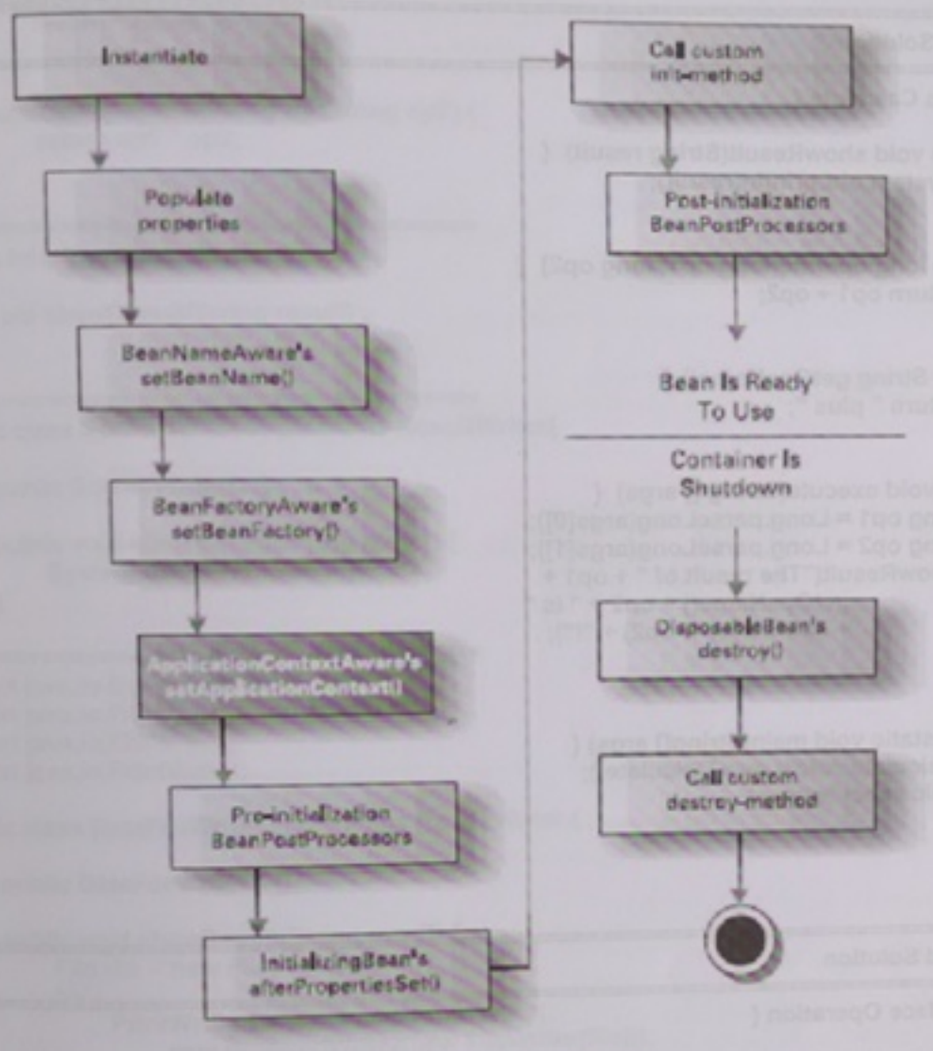


Figure 2.2 The life cycle of a bean in a Spring application context

JPG to PDF Converter For Mac - Unregis

```
==========================================================================
Monolithic Solution
==========================================================================
public class Calculate {

        private void showResult(String result)  {
                System.out.println(result);
        }

        private long operate(long op1, long op2)  {
                return op1 + op2;
        }

        private String getOpsName() {
                return " plus ";
        }

        public void execute(String [] args)  {
                long op1 = Long.parseLong(args[0]);
                long op2 = Long.parseLong(args[1]);
                showResult("The result of " + op1 +
                        getOpsName() + op2 + " is "
                        + operate(op1, op2) + "!");

        }

        public static void main(String[] args) {
                Calculate calc = new Calculate();
                calc.execute(args);
        }
}


==========================================================================
Modularized Solution
==========================================================================
public interface Operation {

        long operate(long op1, long op2);

        String getOpsName();
}
=====================================
public class OpAdd implements Operation {

        public OpAdd() {}

        public String getOpsName() {
                return " plus ";
        }

        public long operate(long op1, long op2) {
                return op1 + op2;
        }

}
=====================================
public class OpMultiply implements Operation {

        public OpMultiply() {}

```


JPG to PDF Converter For Mac - Unregis

```
    public String getOpsName() {
        return " times ";
    }

    public long operate(long op1, long op2) {
        return op1 * op2;
    }
}
=====================================
public interface ResultWriter {

    void showResult(String result) ;

}
=====================================
public class ScreenWriter implements ResultWriter{

    public ScreenWriter() {}

    public void showResult(String result) {
        System.out.println(result);
    }
}
=====================================
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.PrintWriter;

public class DataFileWriter implements ResultWriter {

    public DataFileWriter() {}

    public void showResult(String result) {
        File file = new File("/output.txt");
        try {
            PrintWriter fwriter = new PrintWriter(
                new BufferedWriter(new FileWriter(file)));
            fwriter.println(result);
            fwriter.close();

        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
=====================================
public class CalculateScreen {

    private Operation ops = new OpAdd();
    private ResultWriter wtr = new ScreenWriter();

    public static void main(String[] args) {
        CalculateScreen calc = new CalculateScreen();
        calc.execute(args);
    }

    public void execute(String [] args)  {
        long op1 = Long.parseLong(args[0]);
        long op2 = Long.parseLong(args[1]);
```

JPG to PDF Converter For Mac - Unregis

```
            wtr.showResult("The result of " + op1 +
                        ops.getOpsName() + op2 + " is "
                        + ops.operate(op1, op2) + "!");

        }

}
=====================================
public class CalculateMultFile {

        private Operation ops = new OpMultiply();
        private ResultWriter wtr = new DataFileWriter();

        public static void main(String[] args) {
                CalculateMultFile calc = new CalculateMultFile();
                calc.execute(args);
        }

        public void execute(String [] args)  {
                long op1 = Long.parseLong(args[0]);
                long op2 = Long.parseLong(args[1]);
                wtr.showResult("The result of " + op1 +
                        ops.getOpsName() + op2 + " is "
                        + ops.operate(op1, op2) + "!");

        }
}
=====================================
Spring Solution
=====================================
public interface Operation {

        long operate(long op1, long op2);

        String getOpsName();
}
=====================================
public class OpAdd implements Operation {

        public OpAdd() {}

        public String getOpsName() {
                return " plus ";
        }

        public long operate(long op1, long op2) {
                return op1 + op2;
        }
}
=====================================
public class OpMultiply implements Operation {

        public OpMultiply() {}

        public String getOpsName() {
                return " times ";
        }

        public long operate(long op1, long op2) {
                return op1 * op2;
        }
```


JPG to PDF Converter For Mac - Unregis

```
}
==================================
public interface ResultWriter {

    void showResult(String result) ;

}
==================================
public class ScreenWriter implements ResultWriter{

    public ScreenWriter() {}

    public void showResult(String result) {
        System.out.println(result);
    }
}
==================================
import java.io.*;

public class DataFileWriter implements ResultWriter {

    public DataFileWriter() {}

    public void showResult(String result) {

        File file = new File("/output.txt");

        try
        {
            PrintWriter fwriter = new PrintWriter(
                new BufferedWriter(new FileWriter(file)));
            fwriter.println(result);
            fwriter.close();
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
==================================
public class CalculateSpring
{
    private Operation ops;
    private ResultWriter wtr;

    public void setOps(Operation ops) {
        this.ops = ops;
    }

    public void setWriter(ResultWriter writer) {
        wtr = writer;
    }

    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext(
                "beans.xml");

        BeanFactory factory = (BeanFactory) context;
        CalculateSpring calc =
            (CalculateSpring) factory.getBean("opsbean");
```

JPG to PDF Converter For Mac - Unregis

```
245
246             calc.execute(args);
247         }
248
249         public void execute(String [] args)  {
250
251             long op1 = Long.parseLong(args[0]);
252             long op2 = Long.parseLong(args[1]);
253             wtr.showResult("The result of " + op1 +
254                         ops.getOpsName() + op2 + " is "
255                         + ops.operate(op1, op2) + "!");
256
257         }
258 }
259 ==========================================================================
260 beans.xml
261 ==========================================================================
262 <?xml version="1.0" encoding="UTF-8"?>
263
264 <beans xmlns="http://www.springframework.org/schema/beans"
265        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
266        xsi:schemaLocation="http://www.springframework.org/schema/beans
267              http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
268
269        <bean id = "screen" class = "ScreenWriter" />
270        <bean id = "multiply" class = "OpMultiply" />
271        <bean id = "add" class = "OpAdd" />
272
273        <bean id = "opsbean" class = "CalculateSpring">
274              <property name = "ops" ref = "add" />
275              <property name = "writer" ref = "screen"/>
276        </bean>
277
278 </beans>
279 ==========================================================================
```

JPG to PDF Converter For Mac - Unregis

```
==========================================================================
App 1)>>>>>>>>>> Spring core module application that demonstrates setter injection<<<<<<<<<
==========================================================================
----------------------------------Demo.java-----------------------------
public interface Demo
{
    public String generateWishMsg(String name);
}
----------------------------------DemoBean.java-------------------------
import java.util.*;

public class DemoBean implements Demo
{
    private String msg;

    public void setMsg(String msg)
    {
        this.msg=msg;
    }
    public String generateWishMsg(String name)
    {
      Calendar cl=Calendar.getInstance();
      int h=cl.get(Calendar.HOUR_OF_DAY);

      if(h<=12)
          return msg+"   good morning    :"+name;
      else if(h<=16)
          return msg+" Good afternoon   :"+name;
      else if(h<=20)
          return msg+"   Good Evening    :"+name;
      else
          return msg+"   Good Night    :"+name;
    }
}
----------------------------------DemoCfg.xml---------------------------
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
            "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
  <bean id="db" class="DemoBean">
      <property name="msg">
              <value>hai</value>
      </property>
  </bean>
</beans>
----------------------------------DemoClient.java-----------------------
import org.springframework.core.io.FileSystemResource;
import org.springframework.beans.factory.xml.XmlBeanFactory;


public class DemoClient
{
    public static void main(String args[])
    {
        FileSystemResource res=new FileSystemResource("DemoCfg.xml");
        XmlBeanFactory factory=new XmlBeanFactory(res);
        Object obj=factory.getBean("db");
        DemoInterface bobjref=(DemoInterface)obj;
        System.out.println(bobjref.generateWishMsg("Nataraz"));
    }
}
Note:- jar files in classpath:
```


JPG to PDF Converter For Mac - Unregis

```
1.spring.jar
2.commons-logging.jar

*****************************************************************************
App 2) >>>>>>>>Spring core module application that demonstrates constructor injection<<<<<<
*****************************************************************************
-----------------------------Demo.java-----------------------------

public interface Demo
{
  public String generateWishMsg(String name);
}
-----------------------------DemoBean.java-----------------------------
import java.util.*;

public class DemoBean implements Demo
{
      private String msg;

      public DemoBean(String msg)
      {
            System.out.println("Demobean:1-paramconstructor");
            this.msg=msg;
      }
      public String generateWishMsg(String name)
      {
         Calendar cl=Calendar.getInstance();
         int h=cl.get(Calendar.HOUR_OF_DAY);

        if(h<=12)
            return msg+"  good morning   :"+name;
        else if(h<=16)
            return msg+" Good afternoon  :"+name;
        else if(h<=20)
            return msg+"  Good Evening   :"+name;
        else
            return msg+"  Good Night    :"+name;
      }
}

-----------------------------DemoCfg.xml-----------------------------

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
            "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
  <bean id="db" class="DemoBean">
      <constructor-arg value="raja"/>
  </bean>
</beans>
-----------------------------DemoClient.java-----------------------------
import org.springframework.core.io.FileSystemResource;
import org.springframework.beans.factory.xml.XmlBeanFactory;

public class DemoClient
{
      public static void main(String args[])
      {
            FileSystemResource res=new FileSystemResource("DemoCfg.xml");
            XmlBeanFactory factory=new XmlBeanFactory(res);
```

JPG to PDF Converter For Mac - Unregis

```
            Object obj=factory.getBean("db");
            DemoInterface bobjref=(DemoInterface)obj;
            System.out.println(bobjref.generateWishMsg("rani"));
        }
}
```

Note:- jar files in classpath:

1.spring.jar
2.commons-logging.jar

==========

App 3) Spring core application on injecting one spring bean class object to another spring bean class property(Using <ref> tag)

==========

----------Demo.java----------

```
public interface Demo
{
    public String sayHello();
}
```

----------DemoBean.java----------

```
import java.util.*;

public class DemoBean implements Demo
{
    String name;
    int age;
    Date d;
    TestBean tb;

    public void setName(String name)
    {
        System.out.println("DemoBean:setName()");
        this.name=name;
    }
    public void setAge(int age)
    {
        System.out.println("DemoBean:setAge()");
        this.age=age;
    }
    public void setD(Date d)
    {
        System.out.println("DemoBean:setD()");
        this.d=d;
    }
    public void setTb(TestBean tb)
    {
        System.out.println("DemoBean:setTb()");
        this.tb=tb;
    }
    public String sayHello()
    {
        return "GoodMorning"+"name="+name+"age="+age+"d="+d+"tb="+tb;
    }
}
```

----------DemoCfg.xml----------

```
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
        "http://www.springframework.org/dtd/spring-beans-2.0.dtd">

<beans>
    <bean id="t1" class="TestBean">
        <property name="msg"><value>Hello</value></property>
    </bean>
```

JPG to PDF Converter For Mac - Unregis

```
    <bean id="dt" class="java.util.Date">
        <property name="year"><value>110</value></property>
        <property name="month"><value>12</value></property>
        <property name="date"><value>20</value></property>
    </bean>

    <bean id="db" class="DemoBean">
        <property name="name"><value>Raja</value></property>
        <property name="age"><value>30</value></property>
        <property name="d"><ref bean="dt"/></property>
        <property name="tb"><ref bean="t1"/></property>
    </bean>
</beans>
--------------------------------TestBean.java--------------------------------
public class TestBean
{
    String msg;

    public void setMsg(String msg)
    {
        System.out.println("TestBean:setMsg(-)");
        this.msg=msg;
    }
    public String toString()
    {
        return "TestBean.msg="+msg;
    }
}
--------------------------------DemoClient.java------------------------
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;

public class DemoClient
{
    public static void main(String args[])
    {
        FileSystemResource res=new FileSystemResource("DemoCfg.xml");
        XmlBeanFactory factory=new XmlBeanFactory(res);
        Object obj=factory.getBean("db");
        Demo bobjref=(Demo)obj;
        System.out.println(bobjref.sayHello());
    }
}
Note:- jar files in classpath:
  1.spring.jar
  2.commons-logging.jar

==========================================================================
App 4) >>>>>>>>>>>>>.Spring application on injecting DataSource objects<<<<<<<<<<<<<
==========================================================================
-----------------------------Select.java-----------------------------
public interface Select {

public String fetchName(int eno);
public int fetchSalary(int eno);
}
--------------------------------SelectBean.java--------------------------
import java.sql.Connection;
import java.sql.ResultSet;
```

JPG to PDF Converter For Mac - Unregis

```
import java.sql.Statement;
import javax.sql.DataSource;

public class SelectBean implements Select {

//Bean Property
DataSource ds;
public void setDs(DataSource ds)
{
      this.ds=ds;
      System.out.println("SelectBean:setDs()");
}

public String fetchName(int eno) {
      // TODO Auto-generated method stub
try
 {
      Connection con=ds.getConnection();
      Statement st=con.createStatement();
      ResultSet rs=st.executeQuery("select ename from emp where empno="+eno);
      String name=null;
      if(rs.next())
      {
            name=rs.getString(1);
      }
      rs.close();
      st.close();
      con.close();
      return name;
 }
catch(Exception e)
{
      return null;
 }
}


public int fetchSalary(int eno) {
// TODO Auto-generated method stub
try
{
      Connection con=ds.getConnection();
      Statement st=con.createStatement();
      ResultSet rs=st.executeQuery("select sal from emp where empno="+eno);
      int bsal=0;
      if(rs.next())
      bsal=rs.getInt(1);
      rs.close();
      st.close();
      con.close();
      return bsal;
      }
 catch(Exception e)
  {
   return 0;
  }
}//business method
}//class
-----------------------------SelectCfg.xml-----------------------------
<?xml version="1.0" encoding="UTF-8"?>
<beans
```


JPG to PDF Converter For Mac - Unregis

```
      xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:p="http://www.springframework.org/schema/p"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

  <bean id="drds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
        <property name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>
        <property name="username" value="scott"/>
        <property name="password" value="tiger"/>
   </bean>

   <bean id="sb" class="SelectBean">
        <property name="ds" ref="drds"/>
   </bean>

</beans>
-------------------------------------SelectClient.java-----------------------------
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.FileSystemResource;

public class SelectClient {
public static void main(String args[])
{

   FileSystemResource res=new FileSystemResource("DemoCfg.xml");
   XmlBeanFactory factory=new XmlBeanFactory(res);

   Object obj=ctx.getBean("sb");
   Select bobjref=(Select)obj;

   System.out.println("7900 emp name is"+bobjref.fetchName(7900));
   System.out.println("7900 emp salary is"+bobjref.fetchSalary(7900));

  }
}
Note: jar files in classpath:
   1.spring.jar
   2.commons-logging.jar
   3.ojdbc.jar
==========================================================================
App 5)>>>>>>>>>>>>>>>setter Injection on different types of properties<<<<<<<<<<<<<<<<
==========================================================================
----------------------------Demo.java----------------------------
public interface DemoInterface {
      public String sayHello();

}
---------------------------DemoBean.java---------------------------
public class DemoBean implements Demo {

      String name;
      int age;
      int marks[];
      String subjects[];

      public void setName(String name) {
            this.name = name;
      }
```

JPG to PDF Converter For Mac - Unregis

```
    public void setAge(int age) {
        this.age = age;
    }

    public void setMarks(int[] marks) {
        this.marks = marks;
    }

    public void setSubjects(String[] subjects) {
        this.subjects = subjects;
    }


    public String sayHello() {
    System.out.println("name="+name);
    System.out.println("age="+age);
    System.out.println("marks=");
    for(int i=0;i<marks.length;++i)
    {
    System.out.println(marks[i]+".....");

    }
    System.out.println("subjects=");
    for(int i=0;i<subjects.length;++i)
    {
        System.out.println(subjects[i]+".....");
    }

    return "GOOD MORNING";
  }

}
-----------------------------DemoCfg.xml---------------------
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

  <bean id="db" class="DemoBean">
    <property name="name" value="raja"></property>
    <property name="age" value="22"></property>
    <property name="marks">
      <list>
        <value>90</value>
        <value>95</value>
        <value>98</value>
      </list>
    </property>
    <property name="subjects">
      <list>
        <value>JAVA</value>
        <value>J2EE</value>
        <value>.NET</value>
      </list>
    </property>
  </bean>
```

JPG to PDF Converter For Mac - Unregis

```
</beans>
----------------------------------DemoClient.java----------------------------------
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;

public class DemoClient {
public static void main(String args[])
{
        ClassPathResource res= new ClassPathResource("DemoCfg.xml");
        XmlBeanFactory factory=new XmlBeanFactory(res);
        Demo bref=(Demo)factory.getBean("db");
        System.out.println("Result is:"+bref.sayHello());

  }
}
==========================================================================
App 6)>>>>>>Spring application performs setter injection  on all types of properties<<<<<<<<
==========================================================================
-----------------------------Demo.java----------------------

public interface Demo {
  public String sayHello();

}
-----------------------------DemoBean.java-------------------
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Set;


public class DemoBean implements Demo{
 String name;
 int marks[];
 List names;
 Set phones;
 Map capitals;
 Properties faculties;


public void setName(String name) {
        this.name = name;
}
public void setMarks(int[] marks) {
        this.marks = marks;
}
public void setNames(List names) {
        this.names = names;
}
public void setPhones(Set phones) {
        this.phones = phones;
}
public void setCapitals(Map capitals) {
        this.capitals = capitals;
}
public void setFaculties(Properties faculties) {
        this.faculties = faculties;
}

//implementation of business method
public String sayHello()
```

JPG to PDF Converter For Mac - Unregis

```
{
    System.out.println("name="+name);

    System.out.println("marks=");
    for(int i=0;i<marks.length;i++)
      System.out.println(marks[i]+".....");

    System.out.println("names="+names.toString());
    System.out.println("phones="+phones.toString());
    System.out.println("Capitals="+capitals.toString());
    System.out.println("Faculties="+faculties.toString());

    return "Good morning";
}
}
-------------------------DemoCfg.xml-------------------------

<?xml version="1.0" encoding="UTF-8"?>
<beans
      xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:p="http://www.springframework.org/schema/p"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<bean id="dt" class="DemoBean">
  <property name="name" value="raja"/>
  <property name="marks">
    <list>
      <value>80</value>
      <value>85</value>
      <value>90</value>
    </list>
  </property>

    <property name="names">
      <list>
        <value>Raja</value>
        <value>Rani</value>
        <value>Ram</value>
        <ref bean="dt"/>
      </list>
    </property>
    <property name="phones">
      <set>
        <value>9999999999</value>
        <value>9393939393</value>
        <ref bean="dt"/>
      </set>
    </property>
    <property name="capitals">
      <map>
        <entry>
          <key><value>India</value></key>
          <value>New delhi</value>
        </entry>

        <entry>
          <key><value>Japan</value></key>
          <value>Tokyo</value>
        </entry>
```


JPG to PDF Converter For Mac - Unregis

```
        <entry>
          <key><ref bean="dt"/></key>
          <ref bean="dt"/>
        </entry>
        </map>
    </property>

    <property name="faculties">
    <props>
      <prop key="Nataraz">Jee Faculty</prop>
       <prop key="raja">Spring Faculty</prop>
       <prop key="raz">Struts Faculty</prop>
    </props>
  </bean>
</beans>
--------------------------------DemoClient.java-----------------------------
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;

public class DemoClient {
 public static void main(String args[])
  {
       ClassPathResource res= new ClassPathResource("DemoCfg.xml");
       XmlBeanFactory factory=new XmlBeanFactory(res);
       Demo bref=(Demo)factory.getBean("db");
       System.out.println("Result is:"+bref.sayHello());

    }
}

==========================================================================
App 7)>>>>>.Spring application performs construcor injection  on all types of properties<<<<<
==========================================================================
----------------------------Demo.java--------------------

public interface Demo {
       public String sayHello();

}
----------------------------DemoBean.java--------------
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Set;


public class DemoBean implements Demo{
       int marks[];
       Date d;
       List fruits;
       Set phones;
       Map capitals;
       Properties players;

 public DemoBean(int[] marks, Date d, List fruits, Set phones, Map capitals,
             Properties players) {
       super();
       this.marks = marks;
       this.d = d;
       this.fruits = fruits;
       this.phones = phones;
```

JPG to PDF Converter For Mac - Unregis

```
610         this.capitals = capitals;
611         this.players = players;
612         }
613
614
615   //implementation of business method
616   public String sayHello()
617   {
618
619         System.out.println("marks=");
620         for(int i=0;i<marks.length;i++)
621         System.out.println(marks[i]+".....");
622
623         System.out.println("d="+d);
624         System.out.println("Fruits="+fruits);
625         System.out.println("Phones="+phones);
626         System.out.println("Capitals="+capitals);
627         System.out.println("Players="+players);
628
629         return "Good morning";
630    }
631   }
632   ------------------------DemoCfg.xml--------------------------
633   <?xml version="1.0" encoding="UTF-8"?>
634   <beans
635         xmlns="http://www.springframework.org/schema/beans"
636         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
637         xmlns:p="http://www.springframework.org/schema/p"
638         xsi:schemaLocation="http://www.springframework.org/schema/beans
639          http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
640
641   <bean id="dt" class="java.util.Date"/>
642   <bean id="db" class="DemoBean">
643
644     <constructor-arg>
645        <list>
646           <value>90</value>
647            <value>95</value>
648             <value>100</value>
649        </list>
650     </constructor-arg>
651
652     <constructor-arg><ref bean="dt"/></constructor-arg>
653
654     <constructor-arg>
655        <list>
656           <value>Apple</value>
657           <value>Grapes</value>
658        </list>
659     </constructor-arg>
660
661     <constructor-arg>
662        <set>
663         <value>9999999999</value>
664         <value>9393939393</value>
665        </set>
666     </constructor-arg>
667
668     <constructor-arg>
669        <map>
670           <entry>
```


JPG to PDF Converter For Mac - Unregis

```
        <key><value>AP</value></key>
        <value>Hyderabad</value>
      </entry>

      <entry>
        <key><value>TN</value></key>
        <value>Chennai</value>
      </entry>
    </map>
</constructor-arg>

 <constructor-arg>
    <props>
      <prop key="Sachin">Cricket</prop>
      <prop key="Rana">Shooting</prop>
    </props>
 </constructor-arg>

 </bean>
</beans>
------------------------DemoClient.java-------------------------
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;

public class DemoClient {
public static void main(String args[])
{
      ClassPathResource res= new ClassPathResource("DemoCfg.xml");

      XmlBeanFactory factory=new XmlBeanFactory(res);

      Demo bref=(Demo)factory.getBean("db");
      System.out.println("Result is:"+bref.sayHello());

  }
}
==========================================================================
App 8)>>>>>>>>>>>>>>>>>Spring Application using properties file<<<<<<<<<<<<<<<<<<<<
==========================================================================
--------------------------------myfile.properties-----------------
#jdbc properties
my.driver=oracle.jdbc.driver.OracleDriver
my.url=jdbc:oracle:thin:@localhost:1521:orcl
my.dbuser=scott
my.dbpwd=tiger
-----------------------------------SelectCfg.xml-------------------------
<?xml version="1.0" encoding="UTF-8"?>
<beans
      xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:p="http://www.springframework.org/schema/p"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

  <bean id="propconfig"
  class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
      <property name="location" value="myfile.properties"/>
  </bean>

<bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource"
```

JPG to PDF Converter For Mac - Unregis

```
    <property name="driverClassName" value="${my.driver}"/>
    <property name="url" value="${my.url}"/>
    <property name="username" value="${my.dbuser}"/>
    <property name="password" value="${my.dbpwd}"/>
</bean>

 <bean id="sb" class="SelectBean">
     <property name="ds" ref="drds"/>
 </bean>

</beans>
-------------------Select.java---------------
public interface Select {
    public String fetchName(int eno);
    public int fetchSalary(int eno);
}
--------------------------SelectBean.java-------------------
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import javax.sql.DataSource;


public class SelectBean implements Select {

//Bean Property
DataSource ds;
public void setDs(DataSource ds)
{
      this.ds=ds;
      System.out.println("SelectBean:setDs()");
}

public String fetchName(int eno) {
      try
      {
      Connection con=ds.getConnection();
      Statement st=con.createStatement();
      ResultSet rs=st.executeQuery("select ename from emp where empno="+eno);
      String name=null;
      if(rs.next())
      {
            name=rs.getString(1);
      }
      rs.close();
      st.close();
      con.close();
      return name;
      }
      catch(Exception e)
      {
            return null;
      }
}


public int fetchSalary(int eno) {
  try
  {
      Connection con=ds.getConnection();
      Statement st=con.createStatement();
```


JPG to PDF Converter For Mac - Unregis

```
        ResultSet rs=st.executeQuery("select sal from emp where empno="+eno);
        int bsal=0;
        if(rs.next())
         bsal=rs.getInt(1);
         rs.close();
         st.close();
         con.close();
        return bsal;
  }
 catch(Exception e)
  {
        return 0;
  }
}//business method
}//class
-----------------------------------SelectClient.java-----------------------
import org.springframework.context.support.FileSystemXmlApplicationContext;

public class SelectClient {
 public static void main(String args[])
 {
   FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext("SelectCfg.xml")

    Object obj=ctx.getBean("sb");
    Select bobjref=(Select)obj;

    System.out.println("7900 emp name is"+bobjref.fetchName(7900));
    System.out.println("7900 emp salary is"+bobjref.fetchSalary(7900));

    }
}

==========================================================================
App 9)>>>>>>>>>>>>>>>>>Application on I18N(Internationalization) plain<<<<<<<<<<<<<<<<
==========================================================================
---------------App.properties-----------------
#(BaseFile-->English)
str1 = delete
str2 = save
str3 = stop
str4 = cancel

---------------App.fr_CA.properties---------------
#for french language
str1 = EFFACER
str2 = SAUF
str3 = ARRÊT
str4 = ANNULER

-------------------App_de_DE.properties--------------
#for german language
str1 = LÖSCHEN
str2 = DENN
str3 = ZUG
str4 = ABSAGEN
-----------------App_hi_IN.properties-----------
#for hindhi language
str1=\u0928\u0940\u0916\u093a
str2=\u0936\u0947\u0937\u0930\u0936\u094d\u0915\u093f\u0924\u094d
str3=\u0930\u0947\u0916\u094a
str4=\u0930\u0926\u094d\u0926\u094d
```

JPG to PDF Converter For Mac - Unregis

```
----------------------------I18nApp.java-----------------------------
//I18nApp.java
// internationalization of an application (the captions on buttons will come
//based on the locale specific properties file that is activated)
import java.awt.*;
import javax.swing.*;
import java.util.*;

public class I18nApp {
 public static void main(String args[]){

  // create Locale obj based on given language,country codes
  Locale l= new Locale(args[0],args[1]);

 // picks up properties file based on the Locale obj data
 ResourceBundle r= ResourceBundle.getBundle("App",l);

 JFrame jf = new JFrame();
 Container cp = jf.getContentPane();

 // create btns by getting labels from activated properties file..
 JButton b1 = new JButton(r.getString("str1"));
 JButton b2 = new JButton(r.getString("str2"));
 JButton b3 = new JButton(r.getString("str3"));
 JButton b4 = new JButton(r.getString("str4"));

 cp.setLayout(new FlowLayout());
 cp.add(b1);
 cp.add(b2);
 cp.add(b3);
 cp.add(b4);

 jf.pack();
 jf.setVisible(true);
   }//main
 }//class

 //>javac I18nApp.java
 //>java I18nApp fr CA     (App_fr_CA.properties file will be activated)
 //>java I18nApp hi IN     (App_hi_IN.properties file will be activated)
 //>java I18nApp x y       (App.properties file will be activated(base file)
 ==================================================================<<<<<<<<<<<<<<<<<<<<
 App 10)>>>>>>>>>>>>>>>>>>>>Application On I18N using spring==================
 ==================================================================
 --------------------DemoCfg.xml--------------------
 <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
 "http://www.springframework.org/dtd/spring-beans.dtd">
 <beans>
  <bean id="messageSource"
  class="org.springframework.context.support.ResourceBundleMessageSource">
   <property name="basenames">
    <list>
        <value>App</value>
        <value>App_de_DE</value>
        <value>App_fr_CA</value>
        <value>App_hi_IN</value>
    </list>
   </property>
  </bean>

 </beans>
```


JPG to PDF Converter For Mac - Unregis

```
912  ------------------DemoClient.java----------------------
913  //DemoClient.java
914  import org.springframework.context.support.*;
915  import java.util.*;
916  import javax.swing.*;
917  import java.awt.*;
918
919  public class DemoClient {
920        public static void main(String[] args)throws Exception {
921        //Locale Object
922        Locale l=new Locale(args[0],args[1]);
923       //Activate ApplicationContext container
924          ClassPathXmlApplicationContext ctx=new
             ClassPathXmlApplicationContext("DemoCfg.xml");
925        // get Msgs from proeprties file
926        String msg1=ctx.getMessage("str1",null,"default msg1",l);
927        String msg2=ctx.getMessage("str2",null,"default msg2",l);
928        String msg3=ctx.getMessage("str3",null,"default msg3",l);
929        String msg4=ctx.getMessage("str4",null,"default msg4",l);
930          //Develope Swing Frame
931        JFrame jf = new JFrame();
932        Container cp = jf.getContentPane();
933
934        // create btns by getting labels from activated properties file..
935        JButton b1 = new JButton(msg1);
936        JButton b2 = new JButton(msg2);
937        JButton b3 = new JButton(msg3);
938        JButton b4 = new JButton(msg4);
939
940        cp.setLayout(new FlowLayout());
941        cp.add(b1);
942        cp.add(b2);
943        cp.add(b3);
944        cp.add(b4);
945
946        jf.pack();
947        jf.show();
948        }//main
949  }//class
950  //>javac DemoClient.java
951  //>java DemoClient fr FR        (uses App_fr_FR.properties)
952  //>java DemoClient de DE        (uses App_de_DE.properties)
953  //>java DemoClient x y          (uses App.properties)
954  ==========================================================
955  App 11)>>>>>>>>>>>>>>>>>>>Application on Interface Injection<<<<<<<<<<<<<<<<<
956  ==========================================================
957  >>>>>>>>>>>>>>>>>Example App on Interface injection>>>>>>>>>>>>>>>>>
958  ----------------------Demo.java-------------
959  public interface Demo
960  {
961    public String sayHello();
962  }
963  ---------------------DemoCfg.xml-------------------
964  <!-- DemoCfg.xml (Spring cfg file)  -->
965  <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
966                "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
967
968  <beans>
969   <bean id="db" class="DemoBean" scope="prototype">
970       <property name="msg"><value>hello</value></property>
971   </bean>
```

JPG to PDF Converter For Mac - Unregis

```
  <bean  id="dt" class="java.util.Date"/>
  <bean  id="dt1" class="java.util.Date"/>
</beans>
--------------------DemoBean.java-----------------
// DemoBean.java (Spring Bean -non pojo class)

import org.springframework.context.*;
import org.springframework.beans.factory.*;

public class  DemoBean implements Demo,BeanNameAware,ApplicationContextAware
{
   String msg;
   String bname;
   ApplicationContext ctx;

    //setter injection
    public void setMsg(String msg)
    {
        this.msg=msg;
     }

    // to support interface injection (method of BeanNameAware (I))
      public void setBeanName(String name)
      {
       bname=name;
      }

      // to support Interface injection (method of ApplicationContextAware(I))
       public void setApplicationContext(ApplicationContext ctx)
      {
       this.ctx=ctx;
       }

    public String sayHello()
     {
      System.out.println("logical name current bean is"+bname);
      System.out.println("no.of bean managed by uderlying
      cotainer"+ctx.getBeanDefinitionCount());
      System.out.println("current bean is singleton Bean ?"+ctx.isSingleton(bname));
      System.out.println("current bean is prototype Bean ?"+ctx.isPrototype(bname));

      return "Good Morning";
     }//method
}//class
-----------------------DemoClient.java-----------------
// DemoClient.java (java app as Local Client App )
import org.springframework.core.io.*; //( for FileSystemResource(c) )
import org.springframework.beans.factory.xml.*; //(for XmlBeanFactory (c))
import org.springframework.context.support.*; //(for FileSystemXmlApplicationContext (c))

import java.util.*;

public class DemoClient
{
public static void main(String args[])throws Exception
{
    FileSystemXmlApplicationContext ctx=new
    FileSystemXmlApplicationContext("DemoCfg.xml");
    System.out.println("ApplicationContext container is activated");

    Demo bobj=(Demo)ctx.getBean("db");
```


JPG to PDF Converter For Mac - Unregis

```
    System.out.println("Result is"+bobj.sayHello());

 }//main
}//class
==========================================================================
App 12)>>>>>>>>>>>>>>>>Spring Mini-Project based on Spring Core Module<<<<<<<<<<<<<<<<
==========================================================================
-----------------------------------Main.html-----------------------------------
<frameset rows="30%,*">
    <frame src="Search.jsp">
    <frame name="resultframe">
</frameset>
-----------------------------------Search.jsp-----------------------------------
 <body bgcolor="pink" >

  <form action="controller" target="resultframe">
   <b>Select job</b>
     <select name="job">
         <option value="clerk">CLERKS</option>
         <option value="analyst">ANALYSTS</option>
         <option value="salesman">SALESMEN</option>
         <option value="manager">MANAGERS</option>

     </select>
   <input type="submit" value="search">
  </form>
</body>
<%System.out.println("Serach.jsp"); %>
-----------------------------------------------web.xml-----------------------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
      xmlns="http://java.sun.com/xml/ns/j2ee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
      http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
 <servlet>
  <servlet-name>MainServlet</servlet-name>
  <servlet-class>MainServlet</servlet-class>
 </servlet>

 <servlet-mapping>
  <servlet-name>MainServlet</servlet-name>
  <url-pattern>/controller</url-pattern>
 </servlet-mapping>

</web-app>
-----------------------------------MainServlet.java-----------------------------------
import java.io.IOException;
import java.util.ArrayList;
import javax.servlet.*;
import javax.servlet.http.*;

import org.springframework.beans.factory.xml.*;
import org.springframework.core.io.*;

public class MainServlet extends HttpServlet {

Model mod1=null;


public void init(
```

JPG to PDF Converter For Mac - Unregis

```
{
        System.out.println("Init Method");
        ClasspathResource res=new ClassPathResource("modelconfig.xml");
        XmlBeanFactory factory=new XmlBeanFactory(res);
        mod1=(Model)factory.getBean("mdb");
}//init

public void destroy() {
        System.out.println("destroy()");
        mod1=null;
}

public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        System.out.println("doGet() of MainServlet");
         String job=request.getParameter("job").trim();
        ArrayList al=mod1.search(job);
        request.setAttribute("result",al);

        RequestDispatcher rd=request.getRequestDispatcher("Result.jsp");
         if(rd!=null)
          rd.forward(request,response);
}

public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        System.out.println("doPost() of MainServlet");
        doGet(request,response);
}//doPost

}//MainServlet
------------------------------ Spring Environment-----------------------------------------------------
==========================================================
-----------------------------------------Model.java-----------------------------------------
import java.util.*;
public interface Model {
        public ArrayList search(String desg);
}
-----------------------------------------ModelBean.java-----------------------------------------
import java.util.*;
import javax.sql.*;
import java.sql.*;
import p1.EmpBean;

public class ModelBean implements Model{
        public DataSource ds;

public void setDs(DataSource ds) {
        this.ds = ds;
}
public ArrayList search(String desg)
{
        System.out.println("search() of ModelBean class");
        ArrayList al=new ArrayList();

        try {
         Connection con=ds.getConnection();
         PreparedStatement ps=con.prepareStatement("select * from spring_employee where
         desg=?");
         ps.setString(1,desg);
```


JPG to PDF Converter For Mac - Unregis

```
1152        ResultSet rs=ps.executeQuery();
1153
1154          while(rs.next())
1155          {
1156              EmpBean eb=new EmpBean();
1157              eb.setId(rs.getInt(1));
1158              eb.setName(rs.getString(2));
1159              eb.setDesg(rs.getString(3));
1160              eb.setBsal(rs.getFloat(4));
1161              al.add(eb);
1162            }
1163          }
1164          catch (SQLException e) {
1165              e.printStackTrace();
1166        }
1167
1168        return al;
1169
1170    }
1171  }
1172  ----------------------------------------modelconfig.xml----------------------------------------
1173  <?xml version="1.0" encoding="UTF-8"?>
1174  <!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
      "http://www.springframework.org/dtd/spring-beans.dtd">
1175
1176  <beans>
1177    <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
1178      <property
        name="driverClassName"><value>oracle.jdbc.driver.OracleDriver</value></property>
1179      <property name="url"><value>jdbc:oracle:thin:@localhost:1521:orcl</value></property>
1180      <property name="username"><value>scott</value></property>
1181      <property name="password"><value>tiger</value></property>
1182    </bean>
1183     <bean id="mdb" class="ModelBean" autowire="byName"/>
1184    </beans>
1185    ----------------------------------EmpBean.java----------------------------------------
1186    package p1;
1187
1188    public class EmpBean {
1189
1190        int id;
1191        String name,desg;
1192        float bsal;
1193
1194        public EmpBean()
1195        {
1196          System.out.println("EmpBean Constructor");
1197        }
1198
1199        public float getBsal() {
1200              return bsal;
1201        }
1202        public void setBsal(float bsal) {
1203          this.bsal = bsal;
1204        }
1205        public String getDesg() {
1206              return desg;
1207        }
1208        public void setDesg(String desg) {
1209              this.desg = desg;
1210        }
```

JPG to PDF Converter For Mac - Unregis

```
1211        public int getId() {
1212              return id;
1213        }
1214        public void setId(int id) {
1215              this.id = id;
1216        }
1217        public String getName() {
1218              return name;
1219        }
1220        public void setName(String name) {
1221              this.name = name;
1222        }
1223
1224  }
1225  ================================End of spring Environment=========================
1226  ----------------------------------------------Result.jsp--------------------------
1227  <%@page import="java.util.*,p1.EmpBean"%>
1228  <table>
1229  <% ArrayList list=(ArrayList)request.getAttribute("result");
1230    if(list!=null)
1231    { %>
1232     <tr>
1233       <th>ID</th>
1234       <th>Name</th>
1235       <th>Desg</th>
1236       <th>Salary</th>
1237     </tr>
1238     <%for(int i=0;i<list.size();++i)
1239       {
1240       EmpBean eb=(EmpBean)list.get(i); %>
1241       <tr>
1242          <td><%=eb.getId()%></td>
1243          <td><%=eb.getName()%></td>
1244          <td><%=eb.getDesg()%></td>
1245          <td><%=eb.getBsal()%></td>
1246       </tr>
1247       <% }      %>
1248       </table>
1249      <% } %>
1250
1251      <%System.out.println("Result.jsp"); %>
1252
1253
1254  =======================End of Application============================
1255
1256  Note:jar files required in WEB-INF\lib folder & classpath------------->spring.jar,commons-logging.jar,
      ojdbc14.jar
1257  ----------------------------------------------------------------------------------------------------
1258
1259  ==========================================================================
1260  Spring core module with annotations:
1261  ----------------------------------------
1262  ==================================================
1263  App 13)Spring 2.5 Applicaiton on Annotations (IOC)
1264  ==================================================
1265  -------------Test.java--------------
1266  // Test.java (Spring Interface)
1267  public interface Test
1268  {
1269        public String sayHello();
1270  }
```


JPG to PDF Converter For Mac - Unregis

```
----------------TestBean.java----------
//TestBean.java
import org.springframework.stereotype.*;
import org.springframework.beans.factory.annotation.*;
import java.util.*;

@Service
public class TestBean implements  Test
{
  // Bean property
   Date d;

@Autowired  //to perform auto wiring(by type)  on bean property
public void setD(Date d)
{
      this.d=d;
}

public String sayHello()
{
      return "good morning  "+d.toString();
 }
}
----------------DemoCfg.xml----------
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
          http://www.springframework.org/schema/aop
          http://www.springframework.org/schema/aop/spring-aop-2.5.xsd
          http://www.springframework.org/schema/context
          http://www.springframework.org/schema/context/spring-context-2.5.xsd">

<context:annotation-config />
<!--<context:component-scan base-package="." /> -->

  <bean id="tb" class="TestBean"/>
  <bean id="dt" class="java.util.Date"/>

</beans>
-------------------------TestClient.java----------------
// TestClient.java
import org.springframework.context.support.*;
public class  TestClient
{
   public static void main(String[] args)
   {
       FileSystemXmlApplicationContext ctx=new
       FileSystemXmlApplicationContext("DemoCfg.xml");
       Test beanobj=(Test)ctx.getBean("tb");
       System.out.println("result is "+beanobj.sayHello());

   }//main
}//class
==========================================================
App 14)>>>>>>>>>>>>>>>>>>Spring 3.x Application Using Annotations(IOC)<<<<<<<<<<<<<<<<<<
==========================================================
----------------TestUser.java----------------
package p1;
```

JPG to PDF Converter For Mac - Unregis

```
public interface TestInter
{
    public String sayHello();
}
-----------------------TestBean.java----------
package p1;
 import java.util.Date;
import java.util.List;
import javax.annotation.Resource;
import javax.annotation.Resources;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;
import org.springframework.stereotype.Service;

@Component("tb")
public class TestBean implements TestInter
{
//Bean Property
UserBean u1;

@Value("10")
int no;

Date d1;

@Value("#{T(java.util.Arrays).asList('India','Bharat','Hindustan')}")
String nicknames[];

@Value("#{T(java.util.Arrays).asList('Red','Blue','Green')}")
List colors;


@Resource(name="dfb")
public void setD1(Date d)
{
    d1=d;
}

@Resource(name="ub")
public void setU1(UserBean u1)
{
    this.u1=u1;
}


public String sayHello()
{
 System.out.println("no"+no);
 System.out.println("d1"+d1.toString());
 System.out.println("nicknames");

for(int i=0;i<nicknames.length;i++)
  System.out.println(nicknames[i]);

  System.out.println("Colors="+colors.toString());
  System.out.println("u1="+u1);
  return "Good Morning";
 }
}
-----------------------UserBean.java-----------------
```


JPG to PDF Converter For Mac - Unregis

```
package p1;

import org.springframework.stereotype.*;

@Service("ub")
public class UserBean
{
  String msg="Hello World";

  public String toString()
   {
       return "DemoBean.msg"+msg;
   }
}
--------------DateFactoryBean.java------------
package p1;

import org.springframework.beans.factory.FactoryBean;
import org.springframework.stereotype.Component;

@Component("dfb")
public class DateFactoryBean implements FactoryBean
{
 public Object getObject() throws Exception {
       return new java.util.Date();
  }
  public Class getObjectType() {
       return java.util.Date.class;
   }
  public boolean isSingleton() {
       return false;
  }

}
--------------------TestClient.java-------------
package p1;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.*;

public class TestClient
{
 public static void main(String s[])
  {
       AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext();
       ctx.scan("p1");  //p1 is the package name
       ctx.refresh();
       TestInter test=(TestInter)ctx.getBean("tb");
       System.out.println("result is"+test.sayHello());

  }
}
==========================================================================
>>>>>>>>>>>>Example Applications on Spring Annotations(IOC)<<<<<<<<<<<<<<
==========================================================================
App 15) >>>>>>>>>>>>>on constructor Injection<<<<<<<<<<<<<<<<<<<<<<<<<<<<
==========================================================================
-------------------Demo.java-----------------------------------------------
public interface Demo
{
public String sayHello();
```

JPG to PDF Converter For Mac - Unregis

```
}
--------------------DemoBean.java--------------------
import javax.inject.Inject;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;
@Component("db")
public class DemoBean implements Demo
{  int age;
   String name;
   float avg;

 @Inject
public DemoBean(@Value("30")int age, @Value("Raja")String name, @Value("50.67")float avg) {
   this.age = age;
   this.name = name;
   this.avg = avg;
}
 public String sayHello()
 {
 return "Welcome to Spring age= "+age+" name= "+name+" avg= "+avg;
 }
}
-------------------------DemoClient.java-----------
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class DemoClient {
public static void main(String s[]) throws Exception {
  ApplicationContext ctx = new AnnotationConfigApplicationContext(DemoBean.class);
      //ask spring container to give springbean class object
      Demo d1=(Demo)ctx.getBean("db");
      // calls B.method of SpringBean class
      String result = d1.sayHello();
      System.out.println(result);
  }
}
-------------------------------------------------------------------------------
==============================================================================
App 16)  >>>>>>>>Reading values from properties file(using annotations)<<<<<<<<<<<
==============================================================================
App.properties
-----------------
jdbc.driver=oracle.jdbc.driver.OracleDriver
jdbc.url=jdbc:oracle:thin:@localhost:1521:orcl
db.user=scott
db.pwd=tiger
--------------select.java---------------
package p1;
public interface Select
{
  public void readDBDetails();
}
---------------SelectBean.java--------------
package p1;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.ImportResource;
import org.springframework.stereotype.Component;

@ImportResource("classpath:springCfg.xml")
@Component("sb")
```

JPG to PDF Converter For Mac - Unregis

```
public class SelectBean implements Select
{   private @Value("${db.user}")
    String uname;

    private @Value("${db.pwd}")
    String pwd;

    @Override
    public void readDBDetails() {
       System.out.println("userName="+uname);
       System.out.println("Password="+pwd);
    }
}
---------------------Springcfg.xml---------------------
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:context="http://www.springframework.org/schema/context"
   xsi:schemaLocation="http://cxf.apache.org/core http://cxf.apache.org/schemas/core.xsd
 http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/spring-beans.xsd
 http://www.springframework.org/schema/context
 http://www.springframework.org/schema/context/spring-context.xsd">

   <context:annotation-config/>
   <context:component-scan base-package="p1"/>
   <context:property-placeholder location="./App.properties"/>
   <!-- <context:property-placeholder location="classpath:/App.properties"/> -->
</beans>
-----------------------SelectClient.java-------------------
package p1;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class SelectClient
{
  public static void main(String[] args)
  {
      System.out.println("main():SelectClient");
      ClassPathXmlApplicationContext ctx=new
      ClassPathXmlApplicationContext("SpringCfg.xml");
      Select bobj=(Select)ctx.getBean("sb");
      bobj.readDBDetails();
  }
}
-----------------------------------------------------------
=============================================================
App 17) >>>>>>>custom init-method and custom-destroy methods cfg(annotations)<<<<<<<
=============================================================
---------------Demo.java-----------
package p1;
public interface Demo
{
public String sayHello();
}
---------------DemoBean.java-----------------
package p1;
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;
```

JPG to PDF Converter For Mac - Unregis

```
@Component("db")
public class DemoBean implements Demo {
@Value("durga")
String msg;

@Value("10")
int age;

DemoBean()
{
 System.out.println("DemoBean():constructor");
}

@PostConstruct
public void myInit()
{
System.out.println("Init-Method");
if(age<0){
    age=18;
  System.out.println("Age cannot be -ve,it is set to 18");
  }
}

@PreDestroy
public void myDestroy()
{
  System.out.println("Destroy-method");
  age=0;
}

  public String sayHello() {
      return "Hello :" + msg+"\n Your Age :"+age;
  }

}
------------------------DemoCfg.xml-------------------
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:context="http://www.springframework.org/schema/context"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
      http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-context-2.5.xsd">
       <context:component-scan base-package="p1" />
</beans>
------------------DemoClient.java----------------
package p1;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class DemoClient {
  public static void main(String s[]) throws Exception {

  ClassPathXmlApplicationContext ctx=new ClassPathXmlApplicationContext("DemoCfg.xml");

// ask spring container to give springbean class object
Demo d1 = (Demo) ctx.getBean("db");

// calls B.method of SpringBean class
  String result = d1.sayHello();
  System.out.println(result);

  ctx.close();
```


JPG to PDF Converter For Mac - Unregis

```
}
}
==========>>>>>>>>>>>>>>>>>>>>on Factory Bean(annotations)<<
```

```
    @Resource(name="tb")
    Date  dt;

    public String sayHello()
    {
       return "The time is"+dt;
    }
}
----------------DemoCfg.xml--------------------
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context-2.5.xsd">
  <context:component-scan base-package="p1" />
</beans>
----------------DemoClient.java-----------------
package p1;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class DemoClient
{
   public static void main(String[] args)
   {
      System.out.println("Hello");
      ClassPathXmlApplicationContext ctx=new
      ClassPathXmlApplicationContext("DemoCfg.xml");
      Demo bobj=(Demo)ctx.getBean("db");
      System.out.println(bobj.sayHello());
   }
}
==========
PLAIN JNDI
==========
==========================================================================
App 19 )>>>>>>>>>>.Java application to create InitialContext object representing connectivity
with weblogic registry s/w<<<<<<<<<<<<<
==========================================================================
-----------------------------JndiConnTest.java-------------------------------
import javax.naming.*;
import java.util.*;

public class JndiConnTest
{

public static void main(String args[])throws Exception
   {
      // prepare jndi properties(web logic)
      Hashtable ht=new Hashtable();
      ht.put(Context.INITIAL_CONTEXT_FACTORY,"weblogic.jndi.WLInitialContextFactory");
      ht.put(Context.PROVIDER_URL,"t3://localhost:7001");

      // create InitialContext object
      InitialContext ic=new InitialContext(ht);

      if(ic!=null)
      System.out.println("connection established");
      else
      System.out.println("connection not established");
```


JPG to PDF Converter For Mac - Unregis

```
        NamingEnumeration ne=ic.list("");
        while(ne.hasMore())
        {
            NameClassPair ncp=(NameClassPair)ne.next();
            System.out.println(ncp.getName()+"<------------>"+ncp.getClassName());
        }

        }//main
}//class
==========================================================================
App 20)>>>>>>>>>>>>>>>>>>>Java application to perform "List" operations<<
```

```
 // rebind
  System.out.println("---->rebind");
  ic.rebind("banana",new StringBuffer("green"));

 System.out.println("Listing (after rebind) ");
 NamingEnumeration e1=ic.list("");

while(e1.hasMore())
{
 NameClassPair np=(NameClassPair)e1.next(); // each binding is accessed
 System.out.println(np.getName()+"-----> "+np.getClassName());
} */

 /*  System.out.println("---->unbind operation");
//unbind operation
 ic.unbind("today");

 //listing
 System.out.println("\n\n\nListing---------");
  NamingEnumeration  e2=ic.list("");
 while(e2.hasMore())
 {
  NameClassPair np=(NameClassPair)e2.next();
  System.out.println(np.getName()+" ---> "+np.getClassName());
  }*/

  }//main
}//class
```

==========

App 21)Java application to use the weblogic JDBC connection pool in standalone environmen

==========

----------JndiDsPoolTest.java----------

```
import javax.naming.*;
import java.util.*;
import java.sql.*;
import javax.sql.*;

public class JndiDsPoolTest
{
public static void main(String args[])
{
//prepare Jndi properties
Hashtable ht=new Hashtable();

 ht.put("java.naming.factory.initial","weblogic.jndi.WLInitialContextFactory");
 ht.put("java.naming.provider.url","t3://localhost:7575");

  //create Initial Context
 InitialContext ic=new InitialContext(ht);

 DataSource ds=(DataSource)ic.lookup("DsJndi");

 //write JDBC persistence logic
 Statement st=con.createStatement();

 ResultSet rs=st.executeQuery("select * from student");

 while(rs.next())
 {
  System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
```

JPG to PDF Converter For Mac - Unregis

```
}//while

//close jdbc connections
 rs.close();
 st.close();
 con.close();

}//main
}//class

Note: Add weblogic.jar file to the classpath
==================
SPRING JNDI
==================
=================================================================================
App 22) >>>>>>>>>>>>>>>>>>>>spring based JNDI operations<<<<<<<<<<<<<<<<<<<<<<<<
=================================================================================
-----------------------------------DemoCfg.xml-----------------------------
<?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="jt" class="org.springframework.jndi.JndiTemplate">
    <constructor-arg>
      <props>
        <prop key="java.naming.factory.initial">weblogic.jndi.WLInitialContextFactory</prop>
        <prop key="PROVIDER_URL">t3://localhost:7001</prop>
      </props>
    </constructor-arg>
  </bean>
  <bean id="test" class="JndiTemplateTest">
      <property name="template"><ref bean="jt"/></property>
  </bean>
</beans>

-----------------------------------JndiTemplateTest.java-----------------------------
//DemoClient.java
import org.springframework.context.support.*;
import org.springframework.jndi.*;
import java.util.*;

public class  JndiTemplateTest
{
 static JndiTemplate template;

public void setTemplate(JndiTemplate template)
{
 this.template=template;
}

public static void main(String[] args) throws Exception
{
  FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext("DemoCfg.xml");

  System.out.println("bind()...........");
  template.bind("today",new Date());
  template.bind("apple",new String("it is red"));

  System.out.println("lookup operation");
  Date d1=(Date)template.lookup("today");
  System.out.println("(today)lookup value is"+d1.toString());
```


JPG to PDF Converter For Mac - Unregis

```
template.unbind("today");

/*System.out.println("lookup operation");
d1=(Date)template.lookup("today");
System.out.println("(today)lookup value is"+d1.toString());*/

template.rebind("apple",new String("it is green"));

System.out.println("lookup operation");
String s1=(String)template.lookup("apple");
System.out.println("(apple)lookup value is"+s1.toString());
 }
}
```

→ it leads to exception

→ works fine ✓

jar files in class path
spring.jar
common-logging.jar
weblogic.jar
Keep HTB Domain server of weblogic in running mode ✓

==========

App 23)>>>>>>>>>>>>>>>>>>>>working with JNDI call back interfaces<<<<<<<<<<<<<<<<<<<<

==========

```
----------------------------DemoCfg.xml----------------------------
<?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="jt" class="org.springframework.jndi.JndiTemplate">
    <constructor-arg>
      <props>
       <prop key="java.naming.factory.initial">weblogic.jndi.WLInitialContextFactory</prop>
       <prop key="PROVIDER_URL">t3://localhost:7001</prop>
      </props>
    </constructor-arg>
  </bean>
  <bean id="test" class="JndiTemplateTest">
    <property name="template"><ref bean="jt"/></property>
  </bean>
</beans>
```

→ java.naming.provider.url

→ Jndi properties of weblogic registry ✓

→ class acting as client application is taken as spring bean here ✓

```
----------------------------JndiTemplateTest.java----------------------------
import org.springframework.context.support.*;
import org.springframework.jndi.*;
import java.util.*;

public class  JndiTemplateTest
{
  static JndiTemplate template;

  public void setTemplate(JndiTemplate template)
  {
        this.template=template;
  }
  public static void main(String args[])throws Exception
  {
     FileSystemXmlApplicationContext ctx=new
     FileSystemXmlApplicationContext("DemoCfg.xml");

  //implement list operation.............logic
   template.execute(new JndiCallBack()*)
   {
    public Object doInContext(Context ctx) throws NamingException
    {
        //Use plain JNDI API to implement list operation logic
        NamingEnumeration ne=ctx.list("");

```

→ refer 19

→ setter methods supporting setter injection.

→ method of JndiCallBack interface.

→ represents InitialContext object of Jndi registry ✓

*|→ Here execute method is called having one anonymous innerclass object as argument value. That anonymous innerclass is implementing JndiCallBack interface. and

JPG to PDF Converter For Mac - Unregis

providing implementation for the method of that interface doInContext()


```
1996        while(ne.hasMore())
1997         {
1998            NameClassPair np=(NameClassPair)ne.next();
1999            System.out.println(np.getName()+"-------->"+np.getClassName());
2000         }//while
2001        return null;
2002       }//doInContext(-)
2003     }//inner class
2004    );//execute(-)
2005   }//main
2006  }//class
2007  ==========================================================================
2008  App 24)>>>>>>>>>>>>>>>>>>>>DAO based Spring App <<<<<<<<<<<<<<<<<<<<<<<<
2009  ==========================================================================
2010  ------------------------------Select.java------------------------------------
2011  import java.util.*;
2012
2013  public interface Select
2014  {
2015        public int getSal(int no);
2016        public boolean updateSalary(int no,int newsal);
2017        public boolean fireEmp(int no);
2018        public List getAllEmpDetails()throws Exception;
2019        public Map getEmpDetails(int no)throws Exception;
2020        public boolean registerEmp(int no,String name,String desg,int sal);
2021  }
2022  ------------------------------------SelectBean.java------------------------------------
2023  //SelectBean.java (SpringBean class--- B.logic)
2024
2025  import  java.util.*;
2026
2027  public class SelectBean implements Select
2028  {
2029  EmpDAO  dao;
2030
2031  public void setDao(EmpDAO dao)
2032  {
2033        this.dao=dao;
2034  }
2035
2036  public int getSal(int no)
2037    {
2038        int sal=0;
2039
2040        if(no>0)
2041        sal=dao.fetchSalary(no);
2042        return sal;
2043  }
2044
2045  public boolean updateSalary(int no,int newsal)
2046  {
2047        // b.logic
2048        int sal=dao.fetchSalary(no);
2049
2050        float nsal=sal+(sal*(newsal/100.0f));
2051        System.out.println(nsal);
2052
2053        newsal=Math.round(nsal);
2054        System.out.println(newsal);
2055
2056        // Persistance logic
```

supporting setter method to inject EmpDAO class obj through setter injection

refer line 2166

represents the percentage of hike

JPG to PDF Converter For Mac - Unregis

```
        int res=dao.modifySalary(no,newsal);

        if(res==0)
         return false;
        else
         return true;
 }


 public boolean fireEmp(int no)
 {
        //persistance logic
        int res=dao.deleteRecord(no);

        if(res==0)
         return false;
        else
         return true;
 }

 public List getAllEmpDetails()throws Exception
 {
  List l=dao.getAllRecords();

        if(l!=null)
         return l;
        else
          throw new Exception("no records found");
 }

 public Map getEmpDetails(int no)throws Exception
 {
        Map m=dao.getEmpRecord(no);

        if(m!=null)
         return m;
        else
        throw new Exception("no record found");
 }

 public boolean registerEmp(int no,String name,String desg,int sal)
 {

  int res=dao.insertRecord(no,name,desg,sal);

   if(res==0)
        return false;
   else
        return true;

  }//method
}//class
```

(clean) *1 → shutdown the connection pool when spring container is asked to shutdown) [The dbcp bean cycle

↳ refer 2148

```
----------------------------------SelectCfg.xml----------------------------------
?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="bds" class="org.apache.tomcat.dbcp.dbcp.BasicDataSource"
  destroy-method="close">
   <property name="driverClassName"><value>oracle.jdbc.driver.OracleDriver</value></property>
```

when container before closing bean it will destroy connection pool (custom destroy)

destroy life cycle method (optional to specify here)

commons.

*2 → to work with apache DBCP.

JPG to PDF Converter For Mac - Unregis

gives datasource object representing the jdbc connection pool for oracle.

```
2116  <property name="url"><value>jdbc:oracle:thin:@localhost:1521:orcl</value></property>
2117  <property name="username"><value>scott</value></property>
2118  <property name="password"><value>tiger</value></property>
2119  <property name="initialSize"><value>5</value></property>
2120  </bean>
2121
2122  <bean id="template" class="org.springframework.jdbc.core.JdbcTemplate">
2123      <property name="dataSource"><ref bean="bds"/></property>
2124  </bean>
2125
2126  <bean id="mydao" class="EmpDAO">
2127      <property name="jt"><ref bean="template"/></property>
2128  </bean>
2129
2130  <bean id="sb" class="SelectBean">
2131    <property name="dao">
2132    <ref bean="mydao"/></property>
2133  </bean>
2134  </beans>
```

refer (2114)

our DAO class

refer 2122

our DAO class injected with jdbc template object

refer line no: 2126

Java class implementing DAO design pattern

having Spring DAO persistence logic ✓

our Spring Bean injected with our DAO class object

```
2135  ----------------------------------EmpDAO.java----------------------------------
2136  import org.springframework.jdbc.core.*;
2137  import java.util.*;
2138
2139  public class EmpDAO
2140  {
2141  JdbcTemplate jt;
2142
2143  public void setJt(JdbcTemplate jt)
2144  {
2145      this.jt=jt;
2146  }
2147
2148  public int insertRecord(int no,String name,String desg,int sal)
2149  {
2150
2151   // uses PreparedStatement obj internally
2152   int res=jt.update("insert into Emp (empno,ename,job,sal) values(?,?,?,?)",
2153                     new Object[]{new Integer(no),name,desg,new Integer(sal)});
2154
2155    /*uses PrepredStatement
2156     int res=jt.update("insert into Emp (empno,ename,job,sal) values(?,?,?,?)",
2157                       new Object[]{new Integer(no),name,desg,new Integer(sal)},
2158                       new
                          int[]{Types.INTGER,Types.VARCHAR,Types.VARCHAR,Types.INTGER
2159
2160      /* uses SimpleStatement object
2161      int res=jt.update("insert into Emp(empno,ename,job,sal) values(no,name,desg,sal)"); */
2162
2163      return res;
2164  }
2165
2166  public int fetchSalary(int eno)
2167  {
2168   // uses SimpleStatement object
2169   int sal=jt.queryForInt("select sal from emp where empno="+eno);
2170
2171      /*  uses PreparedStatement obj
2172      int sal=jt.queryForInt("select sal from emp where empno="+eno,new Object[]{}) */
2173
2174      return sal;
2175  }
```

supply parameter values ✓

JPG to PDF Converter For Mac - Unregis

```
public int  modifySalary(int eno,int newsal)
{
  //uses PreparedStaement object
  int res=jt.update("update Emp set  sal=? where empno=?",
                 new Object[]{new Integer(newsal),
                 new Integer(eno)});
     return res;
}


 public int deleteRecord(int eno)
{
     int res=jt.update("delete from Emp where empno=?",new Object[]{new Integer(eno)});
     return res;
}


public Map getEmpRecord(int eno)
{
// uses PrepareStatement object
 Map m=jt.queryForMap("select  *  from emp where empno=?",new Object[]{new Integer(eno)});
 return m;
}

public List getAllRecords()
{
 // uses PrepareStatement object
      List l=jt.queryForList("select  *  from emp",new Object[]{});
      return l;
}

}//class
----------------------------------------SelectClient.java----------------------------------------
import org.springframework.context.support.*;
import java.util.*;

public class SelectClient {
 public static void main(String[] args)throws Exception {

  System.out.println("main method");
  FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext("selectCfg.xml");

   Select bobj=(Select)ctx.getBean("sb");

   // Get Salary (Select)
      System.out.println("Salary is"+bobj.getSal(7499));

   // Update Salary (Update)
      System.out.println("Salary Updated ?"+bobj.updateSalary(7499,10));

   // Delete Employee (Delete)
      System.out.println("Record Deleted ?"+bobj.fireEmp(7876));

  // displays one emp record(Select)
       Map emprecord=bobj.getEmpDetails(7499);
       System.out.println("Emp Details are"+emprecord.get("EMPNO")+
                          emprecord.get("ENAME")+
                          emprecord.get("JOB")+
                          emprecord.get("SAL"));
```

even though if we are using without positional parameters, internally it uses the prepared statement

client application

this is due to the user defined method called throw exception

not the spring method

JPG to PDF Converter For Mac - Unregis

```
// display all emp records (Select)
   List records=bobj.getAllEmpDetails();

System.out.println("All Emp details are");
for(int i=0;i<records.size();++i)
 {
      Map record=(Map)records.get(i);
      System.out.println("Emp Details are"+record.get("EMPNO")+
                         record.get("ENAME")+
                         record.get("JOB")+
                         record.get("SAL"));
 }//for

// Inserting Employee (Insert)
  System.out.println("Employee Inserted ?"+bobj.registerEmp(678,"raja","CLERK",6789,1
}//main

}//class

-----------------------------------------------------------------------------------------
=========================================================================
App 25)>>>>>>>>>>>>>>>>>>>>>(SimpleJdbcTemplateTest.java)<<<<<<<<<<<<<<<<<<<
=========================================================================
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;

public class SimpleJdbcTemplateTest {

  public static void main(String args[]) {

      DriverManagerDataSource dataSource = new DriverManagerDataSource();

      dataSource.setDriverClassName("oracle.jdbc.driver.OracleDriver");
      dataSource.setUrl("jdbc:oracle:thin:@localhost:1521:orcl");
      dataSource.setUsername("scott");
      dataSource.setPassword("tiger");

      JdbcTemplate jt = new JdbcTemplate(dataSource);

      jt.execute("INSERT INTO STUDENT VALUES(900, 'raja1', 'HYD')");
      jt.execute("INSERT INTO STUDENT VALUES(901, 'raja2', 'HYD')");
      jt.execute("INSERT INTO STUDENT VALUES(902, 'raja3', 'HYD')");

      int count = jt.queryForInt("SELECT COUNT(*) FROM STUDENT");
      System.out.println("Count : " + count);

      jt.execute("DELETE FROM STUDENT WHERE SNO >= 900");

      count = jt.queryForInt("SELECT COUNT(*) FROM STUDENT");
      System.out.println("Count : " + count);
  }
}
-----------------------------------------------------------------------------------------
=========================================================================
App 26)>>>>>>>>>>>>>>>>>>>>>(JdbcTemplateSelectTest.java)<<<<<<<<<<<<<<<<<<<
=========================================================================
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
```

↳ for better understanding

JPG to PDF Converter For Mac - Unregis

```
import java.util.ArrayList;
import java.util.List;

import javax.sql.DataSource;


import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.PreparedStatementCreator;
import org.springframework.jdbc.core.PreparedStatementSetter;
import org.springframework.jdbc.core.ResultSetExtractor;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.jdbc.support.rowset.SqlRowSet;
import org.springframework.jdbc.datasource.DriverManagerDataSource;

public class JdbcTemplateSelectTest {

public static void main(String args[]) {

 DriverManagerDataSource dataSource = new DriverManagerDataSource();
 dataSource.setDriverClassName("oracle.jdbc.driver.OracleDriver");
 dataSource.setUrl("jdbc:oracle:thin:@localhost:1521:orcl");
 dataSource.setUsername("scott");
 dataSource.setPassword("tiger");
 JdbcTemplate template = new JdbcTemplate(dataSource);

doSelectOperations(template);
 }

 private static void doSelectOperations(JdbcTemplate template) {

// queryForInt()
int count1 = template.queryForInt("SELECT COUNT(*) FROM EMP");
System.out.println("Total count : " + count1);

// queryForInt() with Parameter
    Object[] parameters1 = new Object[] {"MANAGER"};
    int count2 = template.queryForInt(
     "SELECT COUNT(*) FROM EMP WHERE JOB LIKE ?",parameters1);
     System.out.println("Total count : " + count2);

     // queryForObject()
     Object[] parameters2 = new Object[] {new Integer(7844)};
     Object o = template.queryForObject(
         "SELECT ENAME FROM EMP WHERE EMPNO = ?",
         parameters2,
         String.class);
     System.out.println("Name of 7844 employee is : " + o);


     // Querying for a single domain object.
     Employee emp =
         (Employee) template.queryForObject(
          "SELECT ENAME, JOB, SAL FROM EMP WHERE EMPNO = ?",
          new Object[] {new Integer(7844)},
          new RowMapper() {
            public Object mapRow(ResultSet rs, int rowNum) throws SQLException {
```

not a good practice.

gives datasource object representing jdbc connection pool.

(2330) user defined method call

internally it uses the Statement object (only because we are using only one parameter. If we use two params then it use prep. stmt

uses prepared stat object internally

arg(0) → supplies param values.

arg(0) → query.

arg(2) → String.class → specifies query result type

Put count to see the output in setting values and how you can specify your order (asc/desc)

The one record info will be stored in employee object

JPG to PDF Converter For Mac - Unregis

arg1 → query
arg2 → supplies query param values
arg3 → new RowMapper class object → object of anonymous inner class that implements RowMapper interface having logic to move the current row of resultset to employee class object

```
2359                Employee e = new Employee();
2360
2361                e.setNo(7844);
2362                e.setName(rs.getString("ENAME"));
2363                e.setDesignation(rs.getString(2));
2364                e.setSalary(rs.getDouble(3));
2365
2366                return e;
2367            }//mapRow
2368        });
2369
2370    System.out.println("Details of 7844 employee : " + emp);
2371
2372    // PreparedStatementCreator
2373    final String sql = "SELECT ENAME FROM EMP WHERE DEPTNO = ?";
2374    Object results = template.query(
2375     new PreparedStatementCreator() {
2376         public PreparedStatement createPreparedStatement(Connection con)throws
             SQLException {
2377          PreparedStatement ps = con.prepareStatement(sql);
2378           ps.setInt(1, 10);
2379           return ps;
2380          }
2381         },
2382     new ResultSetExtractor() {
2383
2384     public Object extractData(ResultSet rs)throws SQLException {
2385       List names = new ArrayList();
2386       while(rs.next()){
2387           names.add(rs.getString("ENAME"));
2388       }
2389        return names;
2390     }extractData
2391    });
2392
2393    System.out.println("Employees working in Dept 10 : " + ((ArrayList)results));
2394
2395    // PreparedStatementSetter
2396     String sql1 = "SELECT ENAME FROM EMP WHERE JOB LIKE ?";
2397
2398     Object results1 = template.query(sql1,
2399      new PreparedStatementSetter() {
2400          public void setValues(PreparedStatement ps)throws SQLException {
2401           ps.setString(1, "MANAGER");
2402          }setValues
2403         },
2404      new ResultSetExtractor() {
2405          List names = new ArrayList();
2406          public Object extractData(ResultSet rs)throws SQLException {
2407           while(rs.next()){
2408               names.add(rs.getString("ENAME"));
2409           }
2410
2411           return names;
2412          }//extractData
2413      }
2414     );
2415
2416
2417
2418
```

arg1 anonymous inner class object implementing prepared statement creator interface having logic to create JDBC prepared statement object based on the given connection object.

arg2 object of anonymous inner class that implements resultset extractor interface having logic to copy the record of resultset object to arraylist elements, as employee class object.

arg3 object of anonymous inner class that implements prepared statement setter interface allowing programmer to set values to query parameters using prepared statement object.

arg3 object of anonymous inner class that implements resultset extractor interface having logic to copy resulted records arraylist element.

JPG to PDF Converter For Mac - Unregis

```
System.out.println("All Manager Names : " + ((ArrayList)results1));

// Example of RowSet supported by Spring JDBC.
SqlRowSet rowSet = template.queryForRowSet(
        "SELECT EMPNO,-ENAME, JOB, SAL FROM EMP");
rowSet.last();

while(! rowSet.isBeforeFirst()) {
    int empno = rowSet.getInt(1);
    String ename = rowSet.getString(2);
    String job = rowSet.getString(3);
    double salary = rowSet.getDouble(4);

    System.out.println(empno + "\t" + ename + "\t" +
                job + "\t" + salary);

    rowSet.previous();
  } // end of while loop
  } // main() end of method
} // class
```

→ positing record to last

all(ArrayList obj) names(arraylist)

0 emp obj — String jones
1 emp obj — String BLAKE
2 emp obj — String clark

No class or DTO class

----------Employee.java----------

```
public class Employee {

    private int no;
    private String name;
    private String designation;
    private double salary;

    public Employee() {
        super();
    }

    public int getNo() {
        return no;
    }

    public void setNo(int no) {
        this.no = no;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDesignation() {
        return designation;
    }

    public void setDesignation(String designation) {
        this.designation = designation;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
```

What is the diff b/w preparedStatCreator & preparedStatSetter?

As preparedStatCreator takes the Connection object and from conn object we need to prepare preparedStatement.

But by using preparedStatementSetter, we will get the prepared stat object with the given values.

— ResultSet objects are not serializable objects but cachedRowSet are serializable objects and they can be sent over the network directly.

* → prints the records of table in decrease order.

JPG to PDF Converter For Mac - Unregis

```
            this.salary = salary;
        }

        public String toString() {
            return this.no + "\t"
                + this.name + "\t"
                + this.designation + "\t"
                + this.salary;
        }
}
-----------------------------------------------------------------------
=====================================================================
App 27) >>>>>>>>>>>>>>>>>> NamedParameterTest.java<<
```

```
  <property name="ds"><ref bean="d1"/></property>
</bean>
</beans>

-----------------------------------MyDAO.java-----------------------------
import javax.sql.*;
import org.springframework.core.io.*;
import org.springframework.beans.factory.*;
import org.springframework.beans.factory.xml.*;
import org.springframework.jdbc.core.*;

class  MyDAO
{
      static Resource res=null;
      static BeanFactory factory=null;
      static
      {
      res=new ClassPathResource("selectcfg.xml");
       factory=new XmlBeanFactory(res);
        factory.getBean("m1");
      }

    private static DataSource ds;

      public void setDs(DataSource  ds)
      {
       this.ds=ds;
      }

      public static JdbcTemplate getTemplate()
      {
       JdbcTemplate jt=new JdbcTemplate(ds);
       return jt;
      }
}
---------------------------TestApp.java----------------------------
import org.springframework.jdbc.core.*;
class TestApp
{
      public static void main(String[] args)
      {
      JdbcTemplate jt=MyDAO.getTemplate();
      int res=jt.queryForInt("select count(*) from student");
      System.out.println("count is"+res);
      }
}
-----------------------------------------------------------------------
=====================================================================
App 29)>>>>>>>>> (Calling PL/SQL Procedure using Spring Dao )<<<<<<<<<<<<<<<<<<
=====================================================================
Stored Procedure in ORACLE.
=====================================================================
CREATE OR REPLACE PROCEDURE GET_EMP_DATA
      (NAME IN VARCHAR,
       NO OUT NUMBER,
       DESG OUT VARCHAR,
       SALARY OUT NUMBER)
AS

BEGIN
      SELECT EMPNO, JOB, SAL
```


JPG to PDF Converter For Mac - Unregis

```
        INTO NO, DESG, SALARY
        FROM EMP
        WHERE ENAME = NAME;

END;
/
----------------------------SimpleJdbcCallTest.java----------------------------
import java.sql.Types;
import java.util.HashMap;
import java.util.Map;

import javax.sql.DataSource;

import org.springframework.jdbc.core.SqlOutParameter;
import org.springframework.jdbc.core.SqlParameter;
import org.springframework.jdbc.object.StoredProcedure;
import org.springframework.jdbc.datasource.*;

public class SimpleJdbcCallTest {

    public static void main(String args[]) {

    DriverManagerDataSource dataSource = new DriverManagerDataSource();
      dataSource.setDriverClassName("oracle.jdbc.driver.OracleDriver");
      dataSource.setUrl("jdbc:oracle:thin:@localhost:1521:orcl");
      dataSource.setUsername("scott");
      dataSource.setPassword("tiger");

      Employee emp = (Employee)getEmployeeDetailsWithStoredProcedure(dataSource,
      "ADAMS");
      System.out.println("ADAMS Details : " + emp);
    }

  private static Object getEmployeeDetailsWithStoredProcedure
                          (DataSource ds, String ename) {

      MyStoredProcedure sp = new MyStoredProcedure(ds);
      //call procedure
      Map results = sp.execute(ename);
      //set outparmeter values to emp object
      Employee emp = new Employee();
      emp.setNo((Integer)results.get("NO"));
      emp.setName(ename);
      emp.setDesignation((String)results.get("DESG"));
      emp.setSalary((Double)results.get("SALARY"));

      return emp;
  }

  private static class MyStoredProcedure extends StoredProcedure {

      public MyStoredProcedure(DataSource ds) {

          super(ds, "GET_EMP_DATA");
          this.setFunction(false);
          SqlParameter[] params = {
              new SqlParameter("NAME", Types.VARCHAR),
              new SqlOutParameter("NO", Types.INTEGER),
              new SqlOutParameter("DESG", Types.VARCHAR),
              new SqlOutParameter("SALARY", Types.DOUBLE)
                  };
          this.setParameters(params);
```

JPG to PDF Converter For Mac - Unregis

```
            compile();

        }

        public Map execute(String name) {
            HashMap map = new HashMap();
            map.put("NAME", name);
            return super.execute(map);
        }

    }//inner class
}//outerclass

public class Employee {

    private Integer no;
    private String name;
    private String designation;
    private Double salary;


    public void setNo(Integer no) {
        this.no = no;
    }

    public void setName(String name) {
        this.name = name;
    }


    public void setDesignation(String designation) {
        this.designation = designation;
    }


    public void setSalary(Double salary) {
        this.salary = salary;
    }

    public String toString() {
        return this.no + "\t"
                + this.name + "\t"
                + this.designation + "\t"
                + this.salary;
    }
}
```

Map obj

NO	7874
DESG	clerk
SALARY	9000
keys	values

jar files in class path
spring.jar
commons-logging.jar
ojdbc14.jar

==========Spring DAO mini project==========
==========implementing DAO, MVC2 architectures<<<<<<<<<<<<<<<<
App 30)>>>

```
2722        <bean id="dao" class="MyDao">
2723            <property name="jt"><ref bean="template"/></property>
2724        </bean>
2725        <bean id="mb" class="ModelBean">
2726            <property name="dao"><ref bean="dao"/></property>
2727        </bean>
2728    </beans>
2729    ------------------------MyDao.java------------------------
2730    import org.springframework.jdbc.core.*;
2731    import java.util.*;
2732    public class  MyDao
2733    {
2734        JdbcTemplate jt;
2735
2736    public void setJt(JdbcTemplate jt)
2737    {
2738        this.jt=jt; //setter injection
2739    }
2740
2741    public List getEmpDetails(String condition)
2742    {
2743
2744    // persistance logic
2745        List l=jt.queryForList("select empno,ename,job,sal from emp where job in "+condition+" order
            by job");
2746        return l;
2747    }
2748
2749    }
2750    ------------------------Model.java------------------------
2751    import java.util.*;
2752
2753    public interface Model
2754    {
2755        public List search(String []jobs);
2756    }
2757    ------------------------ModelBean.java------------------------
2758    //ModelBean.java
2759    import java.util.*;
2760
2761    public class  ModelBean implements Model
2762    {
2763        MyDao dao;
2764
2765        public  void  setDao(MyDao dao)
2766        {
2767            this.dao=dao; //setter injection
2768        }
2769
2770        public List search(String[] jobs)
2771        {
2772            // b.logic
2773        StringBuffer sb=new StringBuffer("(");
2774
2775        for (int i=0;i<jobs.length;++i )
2776        {
2777            if(i==(jobs.length-1))
2778                sb.append("'"+jobs[i]+"'");
2779            else
2780                sb.append("'"+jobs[i]+"',");
2781        }//for
```

refer

refer 2722

DAO class (Model layer)

spring Interface (Model layer)

Spring Bean (Model layer)

Deployment Directory Structure

SpringDAOApp
- WEB-INF
- Main.html
- Search.jsp
- result.jsp

Classes
- Model.class
- ModelBean.class
- Model.cfg.txt
- MainServlet.class
- MyDAO.class

JPG to PDF Converter For Mac - Unregis

for utility in class path
spring.jar, servlet-api.jar

for utility in WEB-INF/lib folder
spring.jar, commons-logging.jar, ojdbc14.jar

```
2782
2783      sb.append(")");
2784      String cond=sb.toString();
2785
2786      // use dao's persistance logic
2787       List l=dao.getEmpDetails(cond);
2788
2789       return l;
2790    }//method
2791  }//class
2792  ------------------Main.html----------------------
2793
2794  <frameset rows="30%,*">
2795     <frame src="Search.jsp" name="f1"/>
2796       <frame  name="f2"/>
2797  </frameset>
2798  ---------------------------result.jsp-------------------
2799  <%@ page import="java.util.*;" %>
2800
2801  <% List l=(List) request.getAttribute("result");
2802   if(l!=null)
2803   {  %>
2804    <table border=1>
2805     <tr>
2806     <th>EMPNO</th>
2807     <th>ENAME</th>
2808     <th>SALARY</th>
2809     <th>JOB</th>
2810    </tr>
2811    <%
2812     for(int i=0;i<l.size();++i)
2813     {
2814       Map m=(Map)l.get(i);  %>
2815        <tr>
2816          <td><%=m.get("EMPNO") %></td>
2817          <td><%=m.get("ENAME") %></td>
2818          <td><%=m.get("SAL") %></td>
2819          <td><%=m.get("JOB") %></td>
2820        </tr>
2821        <%} %>
2822    </table>
2823    <%}
2824   else
2825   { %>
2826        <b>No employees are found </b>
2827   <% } %>
2828  ----------------------------------------Search.jsp-------------------
2829
2830  <form target="f2" action="controller" method="get">
2831     <b>select job(s)</b>
2832     <select name="job" multiple>
2833          <option value='CLERK'>clerks</option>
2834          <option value='MANAGER'>managers</option>
2835          <option value='ANALYST'>analyts</option>
2836          <option value='SALESMAN'>salesmen</option>
2837     </select>
2838       <input type="submit" value="search"/>
2839   </form>
2840  ----------------------------MainServlet.java-------------------
2841  import javax.servlet.*;
2842  import javax.servlet.http.*;
```

welcome page (view layer)

result page (view layer)

web.xml
configure main servlet program with /controller url pattern

reading the request attribute value (refer 2868)

NO	name	Sal	job
7499	allen	1600	salesman

loop to display the list data structure as html table row

form page (view layer)

url pattern of main servlet

controller servlet

JPG to PDF Converter For Mac - Unregis

```
import java.io.*;
import java.util.*;

import org.springframework.context.support.*;

public class MainServlet extends HttpServlet
{
Model beanobj=null;

public void init()
{
// activate spring container
 ClassPathXmlApplicationContext ctx=new ClassPathXmlApplicationContext("ModelCfg.xml");
 beanobj=(Model)ctx.getBean("mb");
 }//init

public void doGet(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException
{
// read form data
 String desg[]=req.getParameterValues("job");

      // call B.method
      List l=beanobj.search(desg);

      // keep result in request attribute
      req.setAttribute("result",l);

      // forward the control to result.jsp page
      RequestDispatcher rd=req.getRequestDispatcher("result.jsp");
      rd.forward(req,res);
 }//doGet(-,-)

public void doPost(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException
{
      doGet(req,res);
}//method

public void destroy()
{
      beanobj=null;
}

}//class


==========================================Spring ORM==========================================
App 31)>>>
```

```
{
private HibernateTemplate ht;

 public void setHt(HibernateTemplate ht)
  {
  this.ht=ht;
 }

 public Iterator getData()
 {
  List l=ht.find("from User");
   Iterator it=l.iterator();
   return it;
  }
}
```

logic to inject hibernate template class object through setter injection

orm binding method implementation

HQL (select all records from User table)

returns to caller method

----------SpringHB.xml----------

```
<?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="myds"
     class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName">
      <value>oracle.jdbc.driver.OracleDriver</value>
    </property>
    <property name="url">
      <value>jdbc:oracle:thin:@localhost:1521:orcl</value>
    </property>
    <property name="username">
      <value>scott</value>
    </property>
    <property name="password">
      <value>tiger</value>
    </property>
  </bean>

  <bean id="mySessionFactory"
  class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
   <property name="dataSource" ref="myds"/>
   <property name="mappingResources">
     <list>
       <value>User.hbm.xml</value>
     </list>
   </property>
   <property name="hibernateProperties">
     <props>
       <prop key="hibernate.dialect">org.hibernate.dialect.Oracle9Dialect</prop>
       <prop key="show_sql">true</prop>
     </props>
   </property>
   </bean>

   <bean id="template" class="org.springframework.orm.hibernate3.HibernateTemplate">
     <property name="sessionFactory"><ref bean="mySessionFactory"/></property>
   </bean>

   <bean  id="d1"  class="demoimpl">
     <property name="ht"> <ref bean="template"/> </property>
   </bean>

  </beans>
```

recommended to work with datasource object that points to underlying server managed jdbc connection pool

factorybean that can give hibernate sessionfactory object

ref 2922

hibernate configuration properties

109

provides abstraction layer on plain hibernate programming

refer 2938

refer 2953

spring configuration class configuration

JPG to PDF Converter For Mac - Unregis

```
----------------------------selectservlet.java----------------------
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

import org.springframework.beans.factory.*;
import org.springframework.context.*;
import org.springframework.context.support.*;

public class  selectservlet   extends HttpServlet
{
 public void service(HttpServletRequest request,HttpServletResponse response)
  throws ServletException,IOException
  {    PrintWriter out=response.getWriter();
       System.out.println("In service method of Servlet");

  ApplicationContext ctx=new ClassPathXmlApplicationContext("SpringHB.xml");

  demointer d1=(demointer)ctx.getBean("d1");
  Iterator i1=d1.getData();

  out.println("<body bgcolor=#ffffcc text=red>");
  out.println("<h1><center>all users</h1><hr><br><h3>");
  out.println("<table width=80% border=2>");
  while(i1.hasNext())
  {
       User u1=(User)i1.next();
       out.println("<tr><td>"+u1.getUid()+" <td>"+u1.getUname()+" <td>"+u1.getRole()+"</tr>");
  }
  out.println("</table>");
 }//service(-,-)
}//class
----------------------web.xml----------------------
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
   version="2.4">
   <servlet>
      <servlet-name>select</servlet-name>
      <servlet-class>selectservlet</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>select</servlet-name>
      <url-pattern>/selectaction</url-pattern>
   </servlet-mapping>

</web-app>
----------------------User.java----------------------
public class User
{
private int uid;
private String uname,role;

 public void setUid(int n){uid=n;}
 public int getUid(){ return uid; }
 public void setUname(String s){ uname=s; }
 public String getUname(){ return uname; }
 public void setRole(String r){ role=r; }
```


JPG to PDF Converter For Mac - Unregis

```
3022  public String getRole(){  return role; }
3023 }
3024
3025 /*
3026  create table users
3027  ( userid number(5) primary key,
3028   uname varchar2(20),
3029  role varchar2(20)
3030   );
3031 */
3032 -------------------User.hbm.xml-------------------
3033 <?xml version="1.0"?>
3034 <!DOCTYPE hibernate-mapping PUBLIC
3035 "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
3036 "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
3037 <hibernate-mapping>
3038
3039  <class    name="User"   table="users" >
3040      <id   name="uid"  column="userid"  />
3041      <property   name="uname" />
3042      <property   name="role" />
3043  </class>
3044
3045 </hibernate-mapping>
3046 ==========================================================================
3047 App 32)>>>>>>>>>>>>>>>>>>Spring with hibernate using annotations<<<<<<<<<<<<<<<<<<<<
3048 ==========================================================================
3049 --------------------DemoInter.java-------------------------
3050 package p1;
3051 import java.util.*;
3052 public interface   DemoInter
3053 {
3054     public Iterator getData()throws Exception;
3055 }
3056
3057 ---------------------------------DemoImpl.java---------------------------
3058 package p1;
3059 import org.hibernate.*;
3060 import java.util.*;
3061 import org.springframework.stereotype.*;
3062 import org.springframework.beans.factory.annotation.Autowired;
3063
3064 @Component("d1")
3065 public class DemoImpl implements DemoInter
3066 {
3067 @Autowired
3068  private SessionFactory  sesfact;
3069
3070   public void setSesfact(SessionFactory f1)
3071    {
3072   sesfact=f1;
3073  }
3074
3075   public Iterator getData()throws Exception
3076    {
3077        Session ses=sesfact.openSession();
3078        Query query=ses.createQuery("from  User");
3079     Iterator i1=query.iterate();
3080     return i1;
3081    }
3082 }
```


JPG to PDF Converter For Mac - Unregis

```
----------------------------User.java----------------------------
package p1;
import javax.persistence.*;

@Entity
@Table (name= "users")
public class User
{
  @Id
  @GeneratedValue
  @Column(name = "userid")
  private int uid;

  @Column(name= "uname")
  private String uname;

  @Column (name= "role")
   private String role;

   public void setUid(int n)
   {
     uid=n;
   }

   public int getUid(){ return uid; }
   public void setUname(String s){ uname=s; }
   public String getUname(){ return uname; }
   public void setRole(String r){ role=r; }
   public String getRole(){  return role; }
}
/*
DBTable
-----------
  create table users
  ( userid number(5) primary key,
    uname varchar2(20),
   role varchar2(20)
   );   */

-----------------------------------SpringHB.xml-----------------------------
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
       http://www.springframework.org/schema/context
       http://www.springframework.org/schema/context/spring-context-2.5.xsd">

<context:component-scan base-package="p1" />

  <bean id="myds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName"><value>oracle.jdbc.driver.OracleDriver</value>
    </property>
    <property name="url"> <value>jdbc:oracle:thin:@localhost:1521:orcl</value> </property>
    <property name="username"> <value>scott</value> </property>
    <property name="password"> <value>tiger</value>
    </property>
  </bean>
```

JPG to PDF Converter For Mac - Unregis

```
<bean id="sesfact" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
   <property name="dataSource" ref="myds"/>
      <!-- use this to avoid configuring mapping file-->
   <property name="packagesToScan" value="p1"></property>
 </bean>

</beans>

-----------------------------------------------SelectServlet.java-----------------------------------

package p1;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import org.hibernate.*;
import java.util.*;
import org.springframework.beans.factory.*;
import org.springframework.context.*;
import org.springframework.context.support.*;
public class  SelectServlet   extends HttpServlet
{
 public void service(HttpServletRequest request,HttpServletResponse response)
  throws ServletException,IOException
  {

 PrintWriter out=response.getWriter();
  System.out.println("In service method of Servlet");
  try{
    ApplicationContext ctx=new ClassPathXmlApplicationContext("SpringHB.xml");
    demointer d1=(demointer)ctx.getBean("d1");
    Iterator i1=d1.getData();

   out.println("<body bgcolor=#ffffcc text=red>");
  out.println("<h1><center>all users</h1><hr><br><h3>");
 out.println("<table width=80% border=2>");
  while(i1.hasNext())
  {
      User u1=(User)i1.next();
      out.println("<tr><td>"+u1.getUid()+" <td>"+u1.getUname()+" <td>"+u1.getRole()+"</tr>");
   }
   out.println("</table>");
  }
   catch(Exception e)
   {
      out.println(e);
      e.printStackTrace();
   }
  }//service(-,-)
 }//class
-------------------------------web.xml-------------------

<web-app>
   <servlet>
      <servlet-name>select</servlet-name>
      <servlet-class>p1.SelectServlet</servlet-class>
    </servlet>

    <servlet-mapping>
      <servlet-name>select</servlet-name>
      <url-pattern>/selectaction</url-pattern>
```

JPG to PDF Converter For Mac - Unregis

```
</servlet-mapping>

</web-app>
-----------------------------------------------------

jars in lib
------------
antlr-2.7.7.jar
commons-logging.jar
dom4j-1.6.1.jar
hibernate-commons-annotations-4.0.1.Final.jar
hibernate-core-4.1.8.Final.jar
hibernate-ehcache-4.1.8.Final.jar
hibernate-jpa-2.0-api-1.0.1.Final.jar
javassist-3.15.0-GA.jar
javax.persistence.jar
jboss-logging-3.1.0.GA.jar
jboss-transaction-api_1.1_spec-1.0.0.Final.jar
ojdbc14.jar
org.springframework.asm-3.1.1.RELEASE.jar
org.springframework.aspects-3.1.1.RELEASE.jar
org.springframework.beans-3.1.1.RELEASE.jar
org.springframework.context.support-3.1.1.RELEASE.jar
org.springframework.context-3.1.1.RELEASE.jar
org.springframework.core-3.1.1.RELEASE.jar
org.springframework.expression-3.1.1.RELEASE.jar
org.springframework.jdbc-3.1.1.RELEASE.jar
org.springframework.orm-3.1.1.RELEASE.jar
org.springframework.transaction-3.1.1.RELEASE.jar

==============================Plain Java Mail==============================
App 33) >>>>>>>>>>>>>>>>>>>>>>>>.programs on java mail API<<<<<<<<<<<<<<<<<<<<
=====================================================================
-----------------------------------SendMail.java--------------------
import javax.mail.*;
import javax.mail.internet.*;
import java.util.*;

public class SendMail
{
public static void main(String []args) throws Exception
  {
      //Creating Properties Objct
      Properties p=new Properties();

      //adding protocol,mailserver address & the port number of the mailserver
      p.put("mail.transport.protocol","smtp");
      p.put("mail.smtp.host","localhost");
      p.put("mail.smtp.port","25");

      //Creating the Session Object
      Session session=Session.getInstance(p);

      //Creating and configuring the Message object
      Message msg=new MimeMessage(session);

      msg.setFrom(new InternetAddress("ramesh"));
      InternetAddress address[]={new InternetAddress("ram")};
      msg.setRecipients(Message.RecipientType.TO,address);
      msg.setSentDate(new Date());
      msg.setSubject("open it to know it");
```

JPG to PDF Converter For Mac - Unregis

```
        msg.setText("Hello ! this is the first e-mail from ramesh using Java Mail API on wednesday ");
        //Sending the mail.
        Transport.send(msg);
        System.out.println("mail has been delivered");
    }//main
}//class

================================ReceiveMail.java==============================
/*
  Java Application to receive an E-mail using JAVA Mail API.
  ReceiveMail.java
*/
import javax.mail.*;
import javax.mail.internet.*;
import java.util.*;
public class ReceiveMail
{
  public static void main(String []args) throws Exception
   {
      Properties p=new Properties();
      p.put("mail.transport.protocol","pop3");
      p.put("mail.pop.host","localhost");
      p.put("mail.pop.port","25");

     Session session=Session.getInstance(p);
     Store store=session.getStore("pop3");

     store.connect("localhost","raja","raja");

      Folder folder=store.getDefaultFolder();
     Folder myinbox=folder.getFolder("INBOX");
      myinbox.open(Folder.READ_ONLY);

      System.out.println("No. of messages in the inbox:"+myinbox.getMessageCount());

       Message message=myinbox.getMessage(1);
       message.writeTo(System.out);
       myinbox.close(false);
       store.close();
   }
}
============================DeleteMail.java==================================
/*
  Java Application to Delete  an E-mail using JAVA Mail API.
*/
import javax.mail.*;
import javax.mail.internet.*;
import java.util.*;
public class DeleteMail
{
  public static void main(String []args) throws Exception
   {
      Properties properties=new Properties();
      properties.put("mail.trasport.protocol","pop");
      properties.put("mail.pop.port","110");
      properties.put("mail.host","localhost");

      Session session=Session.getInstance(properties);

      Store store=session.getStore("pop3");

```


JPG to PDF Converter For Mac - Unregis

```
        store.connect("localhost","raja","raja");

        Folder folder=store.getDefaultFolder();
        Folder myinbox=folder.getFolder("INBOX");
        myinbox.open(Folder.READ_WRITE);

        System.out.println("No. of messages in the inbox:"+myinbox.getMessageCount());
        Message message=myinbox.getMessage(1);
        message.setFlag(Flags.Flag.DELETED,true);
        myinbox.close(true);

        System.out.println("No. of messages in the inbox:"+myinbox.getMessageCount());

        store.close();
    }
}

-------------------------------------------------------------------------------------
=====================================================================================
App 35)>>>>>>>>>>>>>>>>Application on Spring Mail<<
```

```
    <property name="protocol"><value>smtp</value></property>
  </bean>

  <bean id="msg" class="org.springframework.mail.SimpleMailMessage">
   <property name="from"><value>rahosh</value></property>
   <property name="subject"><value>open it to know it</value></property>
   <property name="to"><value>nataraj</value></property>
   <property name="text"><value>first mail from spring based java mail</value></property>
  </bean>

  <bean id="sti" class="StudentImpl">
   <property name="mailSender"><ref bean="ms"/></property>
   <property name="message"><ref bean="msg"/></property>
  </bean>
  </beans>
  ==========================MailClient.java==========================
  import org.springframework.beans.factory.*;
  import org.springframework.context.*;
  import org.springframework.context.support.*;
  public class  MailClient
  {
   public static void main(String args[])
    {
   ApplicationContext  ctx=new FileSystemXmlApplicationContext("mailcfg.xml");
    Student bobj=(Student)ctx.getBean("sti");
     bobj.sendEmailMsg();
   System.out.println("mail has been delivered");
    }
  }
  ----------------------------------------
  App 37) >>>>>>>>>Spring Mail with Attachments <<<<<<<<<<<<<<<<<
  ==========================================================
  ----------------------------Student.java----------------------------
  public interface Student {
     public void sendEmailMsg();
  }
  ----------------------------StudentImpl.java----------------------------
  import org.springframework.mail.*;
  import org.springframework.mail.javamail.*;
  import javax.mail.*;
  import javax.mail.internet.*;
  import org.springframework.core.io.*;

  public class  StudentImpl implements Student
  {
   private JavaMailSenderImpl mailSender;

    public void setMailSender(JavaMailSenderImpl mailSender) {
        this.mailSender = mailSender;
    }

   public void sendEmailMsg()
   {
     try{
      mailSender.send(new MimeMessagePreparator() {

       public void prepare(MimeMessage mimeMessage) throws MessagingException {

        MimeMessageHelper message=new MimeMessageHelper(mimeMessage, true, "UTF-8");
        message.setFrom("akshi");
        message.setTo("kalam");
```

mail properties of gmail outgoing server.

allows to create email message

our SpringBean

our bean class object will be injected with gmail sender impl, simple mail message objects

client application

1. Add spring.jar, commons-logging.jar, mail.jar, jfly to class path

call business method (3368)

2. make sure that gmail mail server is in running mode

*1 → Here send method is called with an object of an anonymous inner class that implements MimeMessagePreparator interface

logic to inject objects.

specify the text encoding

To allow multi part content

encoding style

JPG to PDF Converter For Mac - Unregis

```
      message.setSubject("my subject");
      message.setText("hello This mail Content/body");
      message.addAttachment("abc.txt", new ClassPathResource("abc.txt"));
      }
    }); //inner class
   }//try
  catch(MailException me) {
    me.printStackTrace();
  }
 }//sendEmailMsg
}//class
----------------------------------------mailcfg.xml----------------------------------
<?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="ms" class="org.springframework.mail.javamail.JavaMailSenderImpl">
    <property name="host"><value>localhost</value></property>
    <property name="port"><value>25</value></property>
    <property name="protocol"><value>smtp</value></property>
  </bean>

  <bean id="sti" class="StudentImpl">
   <property name="mailSender"><ref bean="ms"/></property>
  </bean>
</beans>
----------------------------------------MailClient.java----------------------------------
import org.springframework.beans.factory.*;
import org.springframework.context.*;
import org.springframework.context.support.*;
public class   MailClient
{
   public static void main(String args[])
    {
      ApplicationContext  ctx=new FileSystemXmlApplicationContext("mailcfg.xml");
      Student bobj=(Student)ctx.getBean("sti");
      bobj.sendEmailMsg();
      System.out.println("mail has been delivered");
    }
}
==========================================
App 36)Prgs on plain java mail api (with Gmail Env.....)
==========================================
----------------------------SendMail.java----------------------------------------
//Mail.java
import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;

public class  SendMail
 {
public SendMail() {
// mail properties outgoing server (gmail.com)
      Properties props = new Properties();
      props.put("mail.smtp.host", "smtp.gmail.com");
      props.put("mail.smtp.port", "465");
      props.put("mail.smtp.auth", "true");
      props.put("mail.smtp.socketFactory.class", "javax.net.ssl.SSLSocketFactory");

      try {
      // create Session obj
```

locates the file related to the attachment

mail properties pointing to the outgoing server of James server

jar files in class path) same as app35

keep James mail server in running mode

mail properties of gmail outgoing server.

JPG to PDF Converter For Mac - Unregis

```
3509        Authenticator auth = new SMTPAuthenticator();
3510
3511        Session session = Session.getInstance(props,auth);
3512
3513        //prepare mail msg
3514        MimeMessage msg = new MimeMessage(session);
3515        //set header values
3516        msg.setSubject("open to it know it");
3517        msg.setFrom(new InternetAddress("nataraz@gmail.com"));
3518        msg.addRecipient(Message.RecipientType.TO, new
            InternetAddress("testemail123@gmail.com"));
3519        //msg text
3520        msg.setText("mail from durga...");
3521
3522        Transport.send(msg);
3523        }//try
3524         catch (Exception ex) {
3525       ex.printStackTrace();
3526         }//catch
3527    }//constructor
3528
3529    private class SMTPAuthenticator extends javax.mail.Authenticator {
3530      public PasswordAuthentication getPasswordAuthentication() {
3531       return new PasswordAuthentication("testmail123@gmail.com","javaj2ee");
3532     }//method
3533    }//inner class
3534    public static void main(String[] args){
3535        SendMail mail=new SendMail();
3536         System.out.println("mail has been delivered");
3537     }//main
3538    }// class
3539    ----------------------------------GetMail.java----------------------------------
3540    import java.util.*;
3541    import javax.mail.*;
3542
3543    public class GetMail {
3544
3545    String host="pop.gmail.com";
3546          String emailfrom="testmail123@gmail.com";
3547          String password="javaj2ee";
3548          String port="995";
3549
3550      GetMail()
3551      {
3552       Properties props = new Properties();
3553
3554          props.put("mail.pop3.host", host);
3555          props.put("mail.pop3.port", port);
3556          props.put("mail.pop3.auth", "true");
3557          props.put("mail.pop3.socketFactory.class", "javax.net.ssl.SSLSocketFactory");
3558
3559        try {
3560          Authenticator auth = new SMTPAuthenticator();
3561          Session session = Session.getInstance(props, auth);
3562
3563           Store store = session.getStore("pop3s");
3564
3565          store.connect(host,emailfrom,password);
3566
3567           Folder folder = store.getFolder("inbox");
3568
```

→ inner class object (refer 3529 to 3533)

→ make sure that mail.jar is added to class path, and also make sure that your machine is connected to Internet ✓

→ mail properties of gmail pronting server ✓

→ connecting to the email account

↳ getting access to the inbox

JPG to PDF Converter For Mac - Unregis

```
        folder.open(Folder.READ_ONLY);
        Message[] msg = folder.getMessages();

        System.out.println((msg.length-1)+" message found");
        for (int i = 1; i <= msg.length; i++) {
            Message mess=folder.getMessage(i);
            mess.writeTo(System.out);
        }//for
    }//try
    catch(Exception e)
    {e.printStackTrace();}
    }//constructor

    private class SMTPAuthenticator extends javax.mail.Authenticator {
     public PasswordAuthentication getPasswordAuthentication() {
        return new PasswordAuthentication(emailfrom, password);
     }
    } //private class

    public static void main(String args[])
    {
     new GetMail();
    }//main
    }//class
    ----------------------------------------------DeleteMail.java----------------------------------------
    import java.util.*;
    import javax.mail.*;

    public class DeleteMail {
        String host="pop.gmail.com";
        String emailfrom="testmail123@gmail.com";
        String password="javaj2ee";
        String port="995";

      DeleteMail()
      {
       Properties props = new Properties();

       props.put("mail.pop3.host", host);
       props.put("mail.pop3.port", port);
       props.put("mail.pop3.auth", "true");
        props.put("mail.pop3.socketFactory.class", "javax.net.ssl.SSLSocketFactory");

        try {
            Authenticator auth = new SMTPAuthenticator();
            Session session = Session.getInstance(props, auth);

            Store store = session.getStore("pop3s");

            store.connect(host,emailfrom,password);

            Folder folder = store.getFolder("inbox");

            folder.open(Folder.READ_WRITE);
            Message[] msg = folder.getMessages();

            msg[0].setFlag(Flags.Flag.DELETED,true); //mark the msg for deletion
            folder.close(true); //close inbox with flag true
            store.close();
            }//try
            catch(Exception e)
```

logic to display all email messages

message index start from '1' (not from '0')

JPG to PDF Converter For Mac - Unregis

```
3630        {e.printStackTrace();}
3631    }//constructor
3632     private class SMTPAuthenticator extends javax.mail.Authenticator {
3633      public PasswordAuthentication getPasswordAuthentication() {
3634          return new PasswordAuthentication(emailfrom, password);
3635     }//method
3636   } //private class
3637
3638   public static void main(String args[])
3639    {
3640     new DeleteMail();
3641    }
3642   }
3643   ==========================================================
3644   ==========================================================================
3645   App 38) >>>>>>>>>>>Sending email message using spring mail to gmail account<<<<<<<<<<
3646   ==========================================================================
3647   ------------------------Student.java------------------
3648   public interface Student {
3649      public void sendEmailMsg();
3650   }
3651   ---------------------------StudentImpl.java-----------------
3652   import org.springframework.mail.MailException;
3653   import org.springframework.mail.MailSender;
3654   import org.springframework.mail.SimpleMailMessage;
3655
3656   public class  StudentImpl implements Student
3657   {
3658     private MailSender mailSender;
3659     private SimpleMailMessage message;
3660
3661      public void setMailSender(MailSender mailSender) {
3662         this.mailSender = mailSender;
3663       }
3664
3665      public void setMessage(SimpleMailMessage message) {
3666         this.message = message;
3667       }
3668    public void sendEmailMsg()
3669    {
3670      try{
3671         mailSender.send(message);
3672       }
3673      catch(MailException me) {
3674         me.printStackTrace();
3675       }
3676    }//sendEmailMsg
3677   }//class
3678   ---------------------------------mailcfg.xml-----------------------------------
3679   <?xml version="1.0" encoding="UTF-8"?>
3680   <!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
3681   "http://www.springframework.org/dtd/spring-beans.dtd">
3682   <beans>
3683     <bean id="ms" class="org.springframework.mail.javamail.JavaMailSender">
3684       <property name="host" value="smtp.gmail.com"/>
3685       <property name="username" value="testmail123@gmail.com"/>
3686        <property name="password" value="javaj2ee"/>
3687        <property name="javaMailProperties">
3688          <props>
3689            <prop key="mail.transport.protocol">smtp</prop>
3690            <prop key="mail.smtp.auth">true</prop>
```

JPG to PDF Converter For Mac - Unregis

```
        <prop key="mail.smtp.SocketFactory.class">javax.net.ssl.SSLSocketFactory</prop>
      </props>
    </bean>

<bean id="msg" class="org.springframework.mail.SimpleMailMessage">
  <property name="from"><value>testmail123@gmail.com</value></property>
  <property name="subject"><value>open it to know it</value></property>
  <property name="to"><value>nataraz@gmail.com</value></property>
  <property name="text"><value>first mail from spring based java mail</value></property>
</bean>

<bean id="sti" class="StudentImpl">
  <property name="mailSender"><ref bean="ms"/></property>
  <property name="message"><ref bean="msg"/></property>
</bean>
</beans>
----------------------------------MailClient.java----------------------------------
import org.springframework.beans.factory.*;
import org.springframework.context.*;
import org.springframework.context.support.*;
public class   MailClient
{
public static void main(String args[])
 {
 ApplicationContext  ctx=new FileSystemXmlApplicationContext("mailcfg.xml");
 BeanFactory  factory=(BeanFactory)ctx;
 Student beanref=(Student)factory.getBean("sti");
  beanref.sendEmailMsg();
 System.out.println("mail has been delivered");
  }
}
-----------------------------------------
==============================Plain RMI==========================
App 39) >>>>>>>>>>>>>>>>>>>>>>>Plain RMI application<<<<<<<<<<<<<<<<<<
=================================================================
----------------------------------Fact.java----------------
import java.rmi.*;

public interface Fact extends Remote
{
 //declaration of business methods
 public long findFactorial(int val)throws RemoteException;
}
----------------------ServerApp.java----------------
import java.rmi.*;
import java.rmi.server.*;

class FactImpl extends UnicastRemoteObject implements Fact
{
public FactImpl() throws RemoteException
{
      super();
      System.out.println("FactImpl:0-param constructor");
}

//implement business method
public long findFactorial(int val)throws RemoteException
{
      System.out.println("FactImpl:findFactorial(-)");
      long res=1;
      for(int i=1;i<=val;++i)
```

mandatory interface

mandatory, (If any problem in n/w cl could get RemoteEx

*1

Business interfa

optional

mandatory

*1 → makes current cl (FactImpl) as regis object in RMI regis

mandatory (only)

JPG to PDF Converter For Mac - Unregis

The object of FactImpl class acts as Business object & Remote object.

```
        {
            res=res*i;
        }//for
        return res;
    }//findFactorial(-)
}//class

public class ServerApp
{
public static void main(String args[])throws Exception
{
  FactImpl bobj=new FactImpl();
```

→ Business object is already here

```
//bind and register business object reference with RMIregistry
  Naming.bind("rmi://localhost:1099/india",bobj);
```

→ business object

protocol

→ nickname/alias name

→ host name and port no. of Rmi registry

```
      System.out.println("server application is ready");

  }//main
}//class
```

----------ClientApp.java----------

```
//ClientApp.java

Import java.rmi.*;

public class ClientApp
{
public static void main(String args[])throws Exception
{
      //Gather business object reference from RMI registry
      Object obj=Naming.lookup("rmi://localhost:1099/india");

      //type casting
      Fact bobj=(Fact)obj;

      //call business methods
      System.out.println("Result is:"+bobj.findFactorial(5));
  }//main
}//class
```

==========

App 40)>>>>>>>>>>Developing spring rmi application<<<<<<<<<

==========

----------first.java----------

```
public interface firstinter
{
 public String wish(String uname);
}
```

----------firstimpl.java----------

```
public class firstimpl implements firstinter
{
  public String wish(String uname)
  {
    return "Good Morning "+uname;
  }
}
```

----------first.xml----------

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
```


JPG to PDF Converter For Mac - Unregis

```
<bean id="f1"  class="firstimpl" />
 <bean id="pfb" class="org.springframework.remoting.rmi.RmiServiceExporter" >
  <property name="service" ref="f1"/>
  <property name="serviceName" value="india"/>
  <property name="serviceInterface" value="firstinter"/>
 </bean>
</beans>
----------------------------firstserver.java----------------------------
import org.springframework.context.*;
import org.springframework.context.support.*;
public class firstserver
{
   public static void main(String args[])
   {
      ApplicationContext ctx=new FileSystemXmlApplicationContext("first.xml");
      System.out.println("server is started..");
   }
}
----------------------------end of server prg----------------------------
==============================clientinter.java==============================
public interface clientinter
{
   public firstinter getInter();
}
==============================clientimpl.java==============================
public class clientimpl implements clientinter
{
   private firstinter firstin;
   public void setFirstin(firstinter f1)
   {
    firstin=f1;
   }
   public firstinter getInter()
    {
    return firstin;
    }
}
==============================firstclient.xml==============================
<?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="pfb" class="org.springframework.remoting.rmi.RmiProxyFactoryBean" >
  <property name="serviceUrl" value="rmi://localhost:1099/india"/>
  <property name="serviceInterface" value="firstinter"/>
 </bean>

 <bean id="c1"  class="clientimpl" >
  <property name="firstin" ref="pfb"/>
  </bean>
</beans>
----------------------------firstclient.java----------------------------
import org.springframework.beans.factory.*;
import org.springframework.context.*;
import org.springframework.context.support.*;
import java.rmi.*;
public class firstclient
{
   public static void main(String args[]) throws RemoteException
   {
   ApplicationContext ctx=new FileSystemXmlApplicationContext("firstclient.xml");
```

Spring Interface is having client side utility method

Spring bean having logic to recieve rmi business object reference and to return to that client application

Supports setter injection

returns to client app (ref 3877)

spring configur

Business Interface

get the business obj reference from rmi registry

ref 3855

the real application that calls the business method of server appl.

to hold rmi business obj reference given by RmiProxyFactoryBean

our spring bean configuration having logic to inject rmi business object reference

JPG to PDF Converter For Mac - Unregis

```
    clientinter i1=(clientinter)ctx.getBean("c1");

    firstinter f1=i1.getInter();

     System.out.println(f1.wish("raj"));
    }
}
```

→ stores ClientImpl object

→ stores rmi business object reference (refer 3845 to 3848)

→ calls the business method of server application

```
----------------------------------------
==========================================================================
App 42)>>>Spring Remoting Application Using Spring DAO in Businness Logic Developement<<<
==========================================================================
  ------ServerSide---------
-----------------------------Salary.java-----------------------------------
public interface Salary
{
  public int  getSalary(String uname);
}
```

→ business method declaration

```
----------------------------SalaryImpl.java----------------------------
import javax.sql.*;
import org.springframework.jdbc.core.*;

public class SalaryImpl  implements  Salary
{
   JdbcTemplate jt;
   public void setJt(JdbcTemplate jt)
   {
       this.jt=jt;
   }


public int getSalary(String uname)
{
  int sal=jt.queryForInt("select  SAL from emp where ename='"+uname+"'");
  return sal;
}
}
```

→ spring DAO based persistance logic that returns employee salary based on given employee name

```
-----------------------------------salary.xml-----------------------------
<!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
  <bean  id="d1" class="org.springframework.jdbc.datasource.DriverManagerDataSource" >
    <property  name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
    <property  name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>
    <property  name="username" value="scott"/>
    <property  name="password" value="tiger/>
   </bean>

   <bean  id="template" class="org.springframework.jdbc.core.JdbcTemplate" >
     <constructor-arg> <ref bean="d1"/></constructor-arg>
   </bean>

   <bean  id="s1"  class="SalaryImpl" >
     <property name="jt"><ref bean="template"/></property>
   </bean>

   <bean  id="pfb" class="org.springframework.remoting.rmi.RmiServiceExporter" >
    <property  name="service" ><ref bean="s1" /></property>
    <property  name="serviceName" ><value>india</value></property>
    <property  name="serviceInterface"> <value>Salary</value></property>
   </bean>
```

→ refer 3915

→ our business class

→ refer 3922

→ to register (s1) bean with rmi registry.

→ refer 3926

JPG to PDF Converter For Mac - Unregis

server application

```
</beans>
----------------------------ServerApp.java----------------------------
import org.springframework.context.*;
import org.springframework.context.support.*;
 public class SalaryServerClient
 {
  public static void main(String args[])
  {
   ApplicationContext ctx=new FileSystemXmlApplicationContext("salary.xml");
   System.out.println("server is started..");
  }
}
 ==========================
 Client code
 ==========================
 -------------------------salaryclient.xml----------------------------
  <?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="pfb" class="org.springframework.remoting.rmi.RmiProxyFactoryBean" >
    <property  name="serviceUrl" value="rmi://localhost:1099/india"/>
    <property name="serviceInterface" value="Salary"/>
  </bean>
  <bean  id="c1"   class="SalaryClient" >
   <property  name="sal" ref="pfb"/>
  </bean>
  </beans>
 -------------------------SalaryClient.java----------------------------
 import org.springframework.beans.factory.*;
 import org.springframework.context.*;
 import org.springframework.context.support.*;
 public class  SalaryClient
 {
  static Salary sal=null;
  public void setSal(Salary sal)
  {
   this.sal=sal;
  }

   public static void main(String args[])
   {
      ApplicationContext ctx=new FileSystemXmlApplicationContext("salaryclient.xml");
      ctx.getBean("c1");
      System.out.println(sal.getSalary("SMITH"));
   }
 }
 ----------------------------------
 ==========================================================================
 App 43)>>>>>>>>>>>>>>>>>>>>>Http Invoker  application<<
```

*3 → bean id of the HttpInvokerServiceExporter pointing to business object (URL U022)


```
{
  public String getWeek()
  {
   String weeks[]=new String[]{"","Sunday","Monday","Tuesday","Wednesday",
                        "Thursday","Friday","Saturday"};
    Calendar  c1=Calendar.getInstance();
    int w=c1.get(Calendar.DAY_OF_WEEK);

    return   "current week :"+weeks[w];
  }
  public String getMonth()
  {
    String months[]=new String[]{"January","February","March","April","May","June","July",
                        "August","September","October","November","December"};
    Calendar  c1=Calendar.getInstance();
    int  m=c1.get(Calendar.MONTH);
    return   "current month :"+months[m];
   }
}
```

→ gives current date and time
→ gives current week day number
→ It gives current month no only from 0 to 11

```
==================================remoting-servlet.xml=====================
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
  <bean  id="h1"   class="httpimpl"/>

  <bean   name="/DemoService"
  class="org.springframework.remoting.httpinvoker.HttpInvokerServiceExporter" >
   <property name="service"><ref  bean="h1"/></property>
    <property  name="serviceInterface"><value>httpinter</value></property>
  </bean>

  <bean id="surl"   class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
     <property name="mappings">
       <props>
         <prop key="my.do">/DemoService</prop>
       </props>
     </property>
  </bean>

</beans>
```

→ spring configuration file
→ DispatcherServlet logical name refer line no: 4044
→ business class configuration as spring bean
→ converts given business object into remote object and keeps in spring container managed registry software
U020
*1
*2
*3
→ service interface name (It becomes nick name or alias name of the registered business object)
→ DD file

```
==================================web.xml=====================
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
   version="2.4">
  <servlet>
   <servlet-name>remoting</servlet-name>
   <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
   <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
   <servlet-name>remoting</servlet-name>
   <url-pattern>*.do</url-pattern>
  </servlet-mapping>
</web-app>
```

→ predefined front controller servlet
→ logical name of the DispatcherServlet
→ extention match url pattern

```
---------------------------------------
========
clientapp
```

*1 → links the client generated requests with the business reference of the registry software.
*2 → It part of client generated URL (proxy)

JPG to PDF Converter For Mac - Unregis

```
========
-----------------------------clientinter.java-----------------------------
public interface   clientinter
{
   public httpinter  getInter();
}
-------------------------clientimpl.java-----------------------------
public class clientimpl implements   clientinter
{ private httpinter  httpin;

  public void setHttpin(httpinter f1)
  {
      httpin=f1;
  }
   public httpinter  getInter()
   {
      return httpin;
   }
}
-------------------------------Demo.xml-----------------------------------------
<?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="pfb"
 class="org.springframework.remoting.httpinvoker.HttpInvokerProxyFactoryBean" >
  <property name="serviceUrl" value="http://localhost:2020/HttpServerApp/my.do"/>
  <property  name="serviceInterface" value="httpinter"/>
 </bean>
 <bean  id="c1"   class="clientimpl" >
  <property  name="httpin" ref bean="pfb"/>
 </bean>
</beans>
-------------------------DemoClient.java-----------------------------------
import org.springframework.beans.factory.*;
import org.springframework.context.*;
import org.springframework.context.support.*;
public class   DemoClient
{
  public static void main(String args[])
   {
   ApplicationContext  ctx=new FileSystemXmlApplicationContext("Demo.xml");
   clientinter c1=(clientinter)ctx.getBean("c1");
   httpinter h1=c1.getInter();
   System.out.println(h1.getWeek());
   System.out.println(h1.getMonth());
   }
}
-----------------------------------------------------------------------------
====================
App 44)Plain JMS
====================
-------------------------MyQueueSender.java-----------------
import javax.jms.*;
import javax.naming.*;
import java.util.Properties;

public class MyQueueSender
{
public static void main(String args[])throws Exception
{
```

JPG to PDF Converter For Mac - Unregis

```
Properties prop=new Properties();

prop.put(Context.INITIAL_CONTEXT_FACTORY,"com.sun.enterprise.naming.SerialInitContextFac
tory");
prop.put(Context.PROVIDER_URL,"iiop://localhost:4848");


InitialContext ic=new InitialContext(prop);
QueueConnectionFactory qcf=(QueueConnectionFactory)ic.lookup("QConFactJndi");
QueueConnection qconn=qcf.createQueueConnection();
QueueSession qsession=qconn.createQueueSession(false,Session.AUTO_ACKNOWLEDGE);

Queue queue=(Queue)ic.lookup("QueueJndi");
QueueSender qsender=qsession.createSender(queue);

    TextMessage message;
    String text;

 int k=1;
  for(int i=1;i<=4;i++)
  {
   if(i==1)
     text=" Hot mail";
   else if(i==2)
     text="Amazon";
   else if(i==3)
     text="JGuru";
   else
     text="Bazee";
   for(int j=1;j<=5;j++)
     {
      message =qsession.createTextMessage();
      message.setStringProperty("Name",text);
      message.setText("This is behaviour"+k+"----------"+text);
      System.out.println("sending"+k+"\t"+text);
      k++;
    }//for
  }//for

  qsession.close();
 qconn.close();
 }
}
----------MyQueueReceiver.java----------
import javax.jms.*;
import javax.naming.*;
import java.util.Properties;


public class MyQueueReceiver implements MessageListener
{
 public static void main(String args[])throws Exception
 {
    Properties prop=new Properties();

    prop.put(Context.INITIAL_CONTEXT_FACTORY,"com.sun.enterprise.naming.SerialInitContextF
    actory");
    prop.put(Context.PROVIDER_URL,"iiop://localhost:4848");


      InitialContext ic=new InitialContext(prop);
```


JPG to PDF Converter For Mac - Unregis

```
        QueueConnectionFactory qcf=(QueueConnectionFactory)ic.lookup("QConFactJndi");
        QueueConnection qconn=qcf.createQueueConnection();
        QueueSession qsession=qconn.createQueueSession(false,Session.AUTO_ACKNOWLEDG

        Queue queue=(Queue)ic.lookup("QueueJndi");
        String selector="Name IN('Bazee','Hotmail')";
        QueueReceiver qreceiver=qsession.createReceiver(queue,selector);
        MyQueueReceiver mq=new MyQueueReceiver();
        qreceiver.setMessageListener(mq);

        qconn.start();
    }//main
    public void onMessage(Message msg)
    {
      try
      {
        TextMessage tmsg=(TextMessage)msg;
        System.out.println("Received Message is"+tmsg.getText());
      }//try
      catch(Exception e)
       {
         e.printStackTrace();
      }
    }
    }
    ----------------TopicDemo.java----------------
    import javax.jms.*;
    import javax.naming.*;
    import java.util.Properties;

    public class TopicDemo
    {
     public static void main(String args[])throws Exception
     {
      Properties prop=new Properties();

      prop.put(Context.INITIAL_CONTEXT_FACTORY,"com.sun.enterprise.naming.SerialInitCont
      ctory");
      prop.put(Context.PROVIDER_URL,"iiop://localhost:4848");
      Context ctx=new InitialContext(prop);
      TopicDemo tt=new TopicDemo();
      tt.doPub(ctx);
      ctx.close();
     }
     public void doPub(Context ctx)throws JMSException,NamingException
      {
          Topic topic=(Topic)ctx.lookup("TopicJndi");

          TopicConnectionFactory tcf=(TopicConnectionFactory)ctx.lookup("TConFactJndi");
          TopicConnection tc=tcf.createTopicConnection();

           TopicSession ts=tc.createTopicSession(false,TopicSession.AUTO_ACKNOWLEDGE);
           TopicPublisher tp=ts.createPublisher(topic);

           tc.start();

           TextMessage msg;
           msg=ts.createTextMessage("This is from my  home");
           System.out.println("Publishing msg on the Destination(Topic)");
```


JPG to PDF Converter For Mac - Unregis

```
        tp.publish(msg);
        tc.close();

    }
}
----------------------------SubsciberDemo.java----------------------------
import javax.jms.*;
import javax.naming.*;
import java.util.Properties;

public class SubscriberDemo
{
  public static void main(String args[])throws NamingException,JMSException
  {
        Properties prop=new Properties();

        prop.put(Context.INITIAL_CONTEXT_FACTORY,"com.sun.enterprise.naming.SerialInitContex
        tFactory");
        prop.put(Context.PROVIDER_URL,"iiop://localhost:4848");
        Context ctx=new InitialContext(prop);

        SubscriberDemo tt=new SubscriberDemo();
        tt.doSub(ctx);
        ctx.close();
  }//main
  public void doSub(Context ctx)throws JMSException,NamingException
  {
        Topic topic=(Topic)ctx.lookup("TopicJndi");

        TopicConnectionFactory tcf=(TopicConnectionFactory)ctx.lookup("TConFactJndi");
        TopicConnection tc=tcf.createTopicConnection();

        TopicSession ts=tc.createTopicSession(false,TopicSession.AUTO_ACKNOWLEDGE);
        TopicSubscriber tsub=ts.createSubscriber(topic);

        tc.start();


        TextMessage msg;
        msg=(TextMessage)tsub.receive();


        System.out.println("Received Message is:"+msg.getText());
        tc.close();
  }//doSub(-)
}//class
----------------------------------
==========================Spring JMS=====================
App 45) >>>>>>>>>>>>>>>On queue<<
```

```
{
    this.jt = jt;
}

public void sendMessage()
{
    MessageCreator creator = new MessageCreator()
    {
    public Message createMessage(Session session)
     {
      TextMessage message = null;
      try
       {
        message = session.createTextMessage();
        message.setText("Durga soft");
        message.setStringProperty("text", "Hello World");
       }
      catch (JMSException e)
       {
        e.printStackTrace();
       }
      return message;
     }
    };
    jt.send(creator);
    }
```

Here the object will be created for the class that implements MessageCreator interface

method of MessageCreator Interface

creates JMS message with body and properties

sends message to queue destination

implements MessageCreator Interface

```
----------------------------spring.xml----------------------
<?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="cf"  class="org.springframework.jndi.JndiObjectFactoryBean">
  <property name="jndiEnvironment">
   <props>
    <prop
    key="java.naming.factory.initial">com.sun.enterprise.naming.SerialInitContextFactory</prop
    <prop key="java.naming.provider.url">iiop://localhost:4343</prop>
   </props>
  </property>
  <property name="jndiName" value="ConFactJndi"/>
 </bean>

 <bean id="q"  class="org.springframework.jndi.JndiObjectFactoryBean">
  <property name="jndiEnvironment">
   <props>
    <prop
    key="java.naming.factory.initial">com.sun.enterprise.naming.SerialInitContextFactory</pro
    p>
    <prop key="java.naming.provider.url">iiop://localhost:4343</prop>
   </props>
  </property>
  <property name="jndiName" value="QueueJndi"/>
 </bean>

 <bean id="template" class="org.springframework.jms.core.JmsTemplate">
  <property name="connectionFactory" ref="cf"/>
  <property name="defaultDestination"  ref="q"/>
 </bean>

 <bean id="MS" class="MessageSender">
```

JPG to PDF Converter For Mac - Unregis

```
  <property name="jt"  ref="template"/>
 </bean>
</beans>
-----------------------SendClient.java-----------------------
import org.springframework.context.support.*;
public class SendClient
{
 public static void main(String args[])
 {
     FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext("spring.xml");
     MessageSender ms =(MessageSender)ctx.getBean("ms");
     ms.sendMessage();
     System.out.println("message sent");
 }
}
---------------------------
ReceiverApp
============================
----------------MessageReceiver.java----------------------
import javax.jms.*;
import org.springframework.jms.core.*;

public class MessageReceiver {
 private JmsTemplate jt;

   public MessageReceiver() {}

   public void setJt(JmsTemplate jt)
   {
      this.jt = jt;
   }

   public void receiveMessage()
   {
      Message message = jt.receive();
      TextMessage tmsg = null;
      if (message instanceof TextMessage)
      {
       tmsg=(TextMessage)message;
        try
        {
         System.out.println(tmsg.getText());
         System.out.println(tmsg.getStringProperty("text"));
        }
        catch (JMSException e)
        {
           e.printStackTrace();
        }
   }//if
 }//method
}
-----------------------------spring.xml-----------------------
<!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>

 <bean id="cf"  class="org.springframework.jndi.JndiObjectFactoryBean">
  <property name="jndiEnvironment">
   <props>
    <prop
    key="java.naming.factory.initial">com.sun.enterprise.naming.SerialInitContextFactory</prop>
```

only spring bean injected with JMS Template class object.

Business method

ReciveApp
- MessageReciever.java/class
- spring.xml
- ReciverClient.java/class

> javac *.java
> java ReciverClient

recieves message from queue destination synchronously.

JPG to PDF Converter For Mac - Unregis

```
    <prop key="java.naming.provider.url">iiop://localhost:4348</prop>
   </props>
  </property>
  <property name="jndiName" value="ConFactJndi"/>
 </bean>

 <bean id="q"  class="org.springframework.jndi.JndiObjectFactoryBean">
  <property name="jndiEnvironment">
   <props>
    <prop
    key="java.naming.factory.initial">com.sun.enterprise.naming.SerialInitContextFactory</pro
    <prop key="java.naming.provider.url">iiop://localhost:4343</prop>
   </props>
  </property>
  <property name="jndiName" value="QueueJndi"/>
 </bean>

 <bean id="template" class="org.springframework.jms.core.JmsTemplate">
   <property name="connectionFactory" ref="cf"/>
   <property name="defaultDestination"  ref="q"/>
 </bean>

 <bean id="ms"  class="MessageReceiver">
   <property name="jt"  ref="template"/>
 </bean>
</beans>
-----------------------ReceiverClient.java-----------------------------
 import org.springframework.context.support.*;
  public class ReceiveClient
  {
    public static void main(String args[])
    {
     FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext("spring.xm
     MessageReceiver mr =(MessageReceiver)ctx.getBean("ms");
        mr.receiveMessage();
        System.out.println("done");
    }
  }
-----------------------------------------
=========================================
App 46) >>>>>>>>>>>>>>>>>>>>>>>>On Topic SenderApp<<<<<<<<<<<<<<<<<<<<<<<<
=========================================================================
---------------------------MessageSender.java-----------------------
import javax.jms.*;
import org.springframework.jms.core.*;
public class MessageSender {

   private JmsTemplate jt;

   public MessageSender() {}

   public void setJt(JmsTemplate jt)
   {
      this.jt = jt;
   }

  public void sendMessage()
   {
   MessageCreator creator = new MessageCreator()
   {
   public Message createMessage(Session session)
```

JPG to PDF Converter For Mac - Unregis

```
    {
     TextMessage message = null;
     try
      {
       message = session.createTextMessage();
       message.setText("DurgaSoft1");
       message.setStringProperty("text", "Hello World");
      }
     catch (JMSException e)
     {
      e.printStackTrace();
      }
      return message;
    }};
    jt.send(creator);
    }
  }
  ---------------------spring.xml--------------------------
  <?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="cf"  class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiEnvironment">
     <props>
       <prop
       key="java.naming.factory.initial">com.sun.enterprise.naming.SerialInitContextFactory</prop
       >
       <prop key="java.naming.provider.url">iiop://localhost:4343</prop>
     </props>
    </property>
    <property name="jndiName" value="ConFactJndi"/>
   </bean>

   <bean id="q"  class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiEnvironment">
     <props>
       <prop
       key="java.naming.factory.initial">com.sun.enterprise.naming.SerialInitContextFactory</prop>
       <prop key="java.naming.provider.url">iiop://localhost:4343</prop>
     </props>
    </property>
    <property name="jndiName" value="TopicJndi"/>
   </bean>

   <bean id="template" class="org.springframework.jms.core.JmsTemplate">
    <property name="connectionFactory" ref="cf"/>
    <property name="defaultDestination"  ref="q"/>
    </bean>

   <bean id="ms"  class="MessageSender">
    <property name="jt"  ref="template"/>
   </bean>
  </beans>
  ----------------------SendClient.java----------------------------
  import org.springframework.context.support.*;
  public class SendClient
  {
  public static void main(String args[])
   {
      FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext("spring.xml");
```


JPG to PDF Converter For Mac - Unregis

```
        MessageSender ms =(MessageSender)ctx.getBean("ms");
        ms.sendMessage();
        System.out.println("message sent");
    }

}
=========
ReceiverApp
=========
----------------------------MessageReceiver.java--------------------------
import javax.jms.*;
import org.springframework.jms.core.*;

public class MessageReceiver {
  private JmsTemplate jt;

    public MessageReceiver() {}

    public void setJt(JmsTemplate jt)
    {
        this.jt = jt;
    }

    public void receiveMessage()
    {
        Message message = jt.receive();
        TextMessage tmsg = null;
        if (message instanceof TextMessage)
        {
         tmsg=(TextMessage)message;
         try
          {
           System.out.println(tmsg.getText());
           System.out.println(tmsg.getStringProperty("text"));
          }
          catch (JMSException e)
          {
            e.printStackTrace();
          }
        }
    }
}
-------------------------spring.xml-------------------------
<!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>

  <bean id="cf"  class="org.springframework.jndi.JndiObjectFactoryBean">
   <property name="jndiEnvironment">
    <props>
     <prop
     key="java.naming.factory.initial">com.sun.enterprise.naming.SerialInitContextFactory</prop>
     <prop key="java.naming.provider.url">iiop://localhost:4343</prop>
    </props>
   </property>
   <property name="jndiName" value="ConFactJndi"/>
  </bean>

  <bean id="jc"  class="org.springframework.jndi.JndiObjectFactoryBean">
   <property name="jndiEnvironment">
    <props>
```

JPG to PDF Converter For Mac - Unregis

```
    <prop
    key="java.naming.factory.initial">com.sun.enterprise.naming.SerialInitContextFactory</prop>
    <prop key="java.naming.provider.url">iiop://localhost:4343</prop>
    </props>
  </property>
  <property name="jndiName" value="TopicJndi"/>
 </bean>

 <bean id="template" class="org.springframework.jms.core.JmsTemplate">
   <property name="connectionFactory" ref="cf"/>
   <property name="defaultDestination"  ref="q"/>
 </bean>

 <bean id="ms"  class="MessageReceiver">
   <property name="jt"  ref="template"/>
 </bean>
</beans>
-----------------------------------ReceiveClient  .java-----------------------
import org.springframework.context.support.*;
public class ReceiveClient
{
 public static void main(String args[])
  {
   FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext("spring.xml");
       MessageReceiver mr =(MessageReceiver)ctx.getBean("ms");
       mr.receiveMessage();
       System.out.println("done");
  }
}
-----------------------------------------------------------
==========================================================================
App 47)>>>>>>>>>>>>>>>>>>>>>>>>Spring  webservices application<<<<<<<<<<<<<<<<<<<<<<<<
==========================================================================
==========
ServerApp
=========
-----------------------------------CalculatorInter.java-----------------------
package com.ds;
public interface CalculatorInter {
      public String sum(int x,int y);

}
-----------------------------------CalculatorImpl.java-----------------------
package com.ds;

import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import org.springframework.stereotype.Service;

@SOAPBinding(style=SOAPBinding.Style.RPC, use=SOAPBinding.Use.LITERAL,
parameterStyle=SOAPBinding.ParameterStyle.WRAPPED)
@WebService(serviceName="CalcyWs")
@Service("Calcy")
public class CalculatorImpl implements CalculatorInter {
      @WebMethod
public String sum(int x, int y) {
      return ""+(x+y);
      }

}
```

Business Interface

Declaration of business method.

plain jax-ws based api

spring-api

plain jax-ws-api

optional

mandatory → It makes the current class as web service component

makes current class as spring bean.

If we don't mention @webservice annotation then Calculator Impl Service is the web service name

JPG to PDF Converter For Mac - Unregis

```
----------------------------------servercfg.xml----------------------------
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:p="http://www.springframework.org/schema/p"
 xmlns:aop="http://www.springframework.org/schema/aop"
 xmlns:context="http://www.springframework.org/schema/context"
 xmlns:jee="http://www.springframework.org/schema/jee"
 xmlns:tx="http://www.springframework.org/schema/tx"
 xmlns:task="http://www.springframework.org/schema/task"
 xmlns:task="http://www.springframework.org/schema/aop
 xsi:schemaLocation="http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
 http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
   http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
   http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context-3.0.xsd
   http://www.springframework.org/schema/jee
   http://www.springframework.org/schema/jee/spring-jee-3.0.xsd
   http://www.springframework.org/schema/tx
   http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
   http://www.springframework.org/schema/task
   http://www.springframework.org/schema/task/spring-task-3.0.xsd">
 <context:component-scan base-package="com.ds"/>
    <bean class="org.springframework.remoting.jaxws.SimplejaxWsServiceExporter">
        <property name="baseAddress" value="http://localhost:5678"/>
    </bean>
 </beans>
 ----------------------------ServiceExposer.java-----------------------
package com.ds;
import org.springframework.context.support.*;
public class ServiceExposer {
  public static void main(String[] args)
  {

   FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext("ServerCfg.xml");
   System.out.println("Service Exposed successfully");

  }
}

/* to access wsdl type the following
http://localhost:5678/CalcyWs?WSDL
*/
//jars needed
/*
org.sf.context.jar,
beans.jar,
core.jar,
commons-logging.jar,-------->collect from spring3
asm.jar,
expression.jar,
web.jar
*/
==========
ClientApp
==========
----------------------------ClientCfg.xml----------------------------
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:p="http://www.springframework.org/schema/p"
 xmlns:aop="http://www.springframework.org/schema/aop"
```


JPG to PDF Converter For Mac - Unregis

```
xmlns:context="http://www.springframework.org/schema/context"
xmlns:jee="http://www.springframework.org/schema/jee"
xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:task="http://www.springframework.org/schema/task"
xsi:schemaLocation="http://www.springframework.org/schema/task"
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
 http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
 http://www.springframework.org/schema/context
 http://www.springframework.org/schema/context/spring-context-3.0.xsd
 http://www.springframework.org/schema/jee
 http://www.springframework.org/schema/jee/spring-jee-3.0.xsd
 http://www.springframework.org/schema/tx
 http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
 http://www.springframework.org/schema/task
 http://www.springframework.org/schema/task/spring-task-3.0.xsd">

  <bean id="gs" class="org.springframework.remoting.jaxws.JaxWsPortProxyFactoryBean">
    <property name="wsdlDocumentUrl" value="http://localhost:5678/CalcyWs?WSDL"/>
    <property  name="serviceName" value="CalcyWs"/>
    <property name="namespaceUri" value="http://ds.com/"/>
    <property name="serviceInterface" value="p1.CalculatorInter"/>
    <property name="portName" value="CalculatorImplPort"/>
  </bean>
</beans>
----------------------------------------ServiceClient.java----------------------------------------
package p1;
import org.springframework.context.support.FileSystemXmlApplicationContext;
public class ServiceClient {
public static void main(String[] args) throws Exception
{
 FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext("ClientCfg.xml");
 CalculatorInter obj=(CalculatorInter)ctx.getBean("gs");
 System.out.println(obj.sum(15, 10));
 }
}
/*
jar files needed(collect from spring3 soft
****************
context.jar,
beans.jar
core.jar
asm.jar
expression.jar
web.jar
aop.jar
*********Collect separtely
commons-logging.jar(from spring2)
aopalliance-1.0.jar (internet)
*/
==========>>>>>>>>>>>>>>Timer Application using core java<<
```

```
4750    {
4751     System.out.println("--------------------");
4752     System.out.println("welcome to durga ");
4753     System.out.println("--------------------");
4754     System.out.println("");
4755    System.out.println("");
4756     }
4757  }
4758  ----------------------------scheduledemo.java----------------------
4759  import java.util.*;
4760  public class scheduledemo
4761  {
4762   public static void main(String args[])
4763   {
4764   Timer t1=new Timer();
4765  //t1.schedule(new task1(),5000);
4766
4767   //t1.schedule(new task1(),5000,3000);
4768
4769   Calendar c1=Calendar.getInstance();
4770   c1.set(2005,Calendar.AUGUST,5,12,04,20);
4771  t1.schedule(new task1(),c1.getTime());
4772   }
4773  }
4774  ------------------------------------------------------------------------------------------------------
4775  ==========================================================================
4776  App 49) >>>>>>>>>>>>>>>>>>>>>>>Timer Appliation Using Spring<<<<<<<<<<<<<<<<<<<
4777  ==========================================================================
4778  -------------------------task1.java-------------------------------
4779  import java.util.*;
4780  public class  task1 extends TimerTask
4781  {
4782     public void run()
4783     {
4784        System.out.println("--------------------");
4785        System.out.println("welcome to durga");
4786        System.out.println("--------------------");
4787        System.out.println("");
4788        System.out.println("");
4789     }
4790  }
4791  -----------------------task2.java-------------------------------
4792  import java.util.*;
4793  import java.io.*;
4794  public class  task2  extends TimerTask
4795  {
4796   public void run()
4797    { try{
4798        FileWriter  fp=new FileWriter("demofile.txt",true);
4799        fp.write(new Date().toString()+"\n");
4800        fp.close();
4801        System.out.println("--------------------");
4802        System.out.println("data is stored");
4803        System.out.println("--------------------");
4804        System.out.println("");
4805        System.out.println("");
4806        }catch(Exception e){}
4807     }
4808  }
4809  -------------------------Timer.xml-------------------------------
4810  <?xml version="1.0" encoding="UTF-8"?>
```


JPG to PDF Converter For Mac - Unregis

```
<!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
<bean  id="job1"  class="task1" />
<bean  id="job2"  class="task2" />
 <bean  id="schedule1"  class="org.springframework.scheduling.timer.ScheduledTimerTask" >
  <property name="delay"><value>5000</value></property>
  <property name="period"><value>2000</value></property>
  <property name="timerTask" ><ref bean="job1" /></property>
</bean>

<bean  id="schedule2"  class="org.springframework.scheduling.timer.ScheduledTimerTask" >
  <property name="delay"> <value>5000</value></property>
  <property name="period"><value>3000</value></property>
  <property name="timerTask" ><ref bean="job2" /></property>
</bean>

<bean   id="stt"  class="org.springframework.scheduling.timer.TimerFactoryBean">
 <property name="scheduledTimerTasks">
  <list>
    <ref  bean="schedule1" />
    <ref  bean="schedule2" />
  </list>
 </property>
 </bean>

</beans>
----------------------------------scheduledemo.java--------------------------------
import org.springframework.context.*;
import org.springframework.context.support.*;
import java.io.*;
public class   scheduledemo
{
 public static void main(String args[]) throws IOException
   {
  ApplicationContext  ctx=new FileSystemXmlApplicationContext("Timer.xml");
  System.in.read();
   }
}
-----------------------------------------------------------------------------------------
==========================================================================<<<<<<
App 50) >>>>>>Spring Based Timer Appliction (Applying Timer on user-defined method)
==========================================================================
-------------------------------task1.java-------------------------------
public class  task1
{
   public void  method1()
    {
        System.out.println("-------------------");
        System.out.println("welcome to Durga");
        System.out.println("-------------------");
        System.out.println("");
        System.out.println("");
    }
}
-----------------------------task2.java-----------------------------
import java.io.*;
import java.util.*;
public class  task2
{
   public void method2()
```

JPG to PDF Converter For Mac - Unregis

```
    {
     try{
        FileWriter  fp=new FileWriter("demofile.txt",true);
        fp.write(new Date().toString()+"\n");
        fp.close();
        System.out.println("--------------------");
        System.out.println("data is stored");
        System.out.println("--------------------");
        System.out.println("");
        System.out.println("");
        }catch(Exception e){}
     }
}
--------------------------------Timer.xml--------------------------------
<?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="job1"  class="task1" />
    <bean  id="job2"  class="task2" />

 <bean id="mt1"
 class="org.springframework.scheduling.timer.MethodInvokingTimerTaskFactoryBean">
   <property  name="targetObject"><ref bean="job1" /></property>
   <property  name="targetMethod"><value>method1</value> </property>
</bean>

<bean id="mt2"
class="org.springframework.scheduling.timer.MethodInvokingTimerTaskFactoryBean">
   <property  name="targetObject"><ref bean="job2" /> </property>
   <property  name="targetMethod"><value>method2</value></property>
</bean>

 <bean  id="schedule1"  class="org.springframework.scheduling.timer.ScheduledTimerTask" >
  <property name="delay"> <value>5000</value></property>
  <property name="period"><value>2000</value></property>
  <property name="timerTask" ><ref bean="mt1" /></property>
</bean>

<bean  id="schedule2"  class="org.springframework.scheduling.timer.ScheduledTimerTask" >
 <property name="delay"> <value>5000</value></property>
 <property name="period"><value>3000</value></property>
 <property name="timerTask" ><ref bean="mt2" /></property>
</bean>

<bean   id="stt"  class="org.springframework.scheduling.timer.TimerFactoryBean">
  <property name="scheduledTimerTasks">
   <list>
     <ref  bean="schedule1" />
     <ref  bean="schedule2" />
   </list>
  </property>
 </bean>
</beans>
--------------------------------scheduledemo.java--------------------------------
import org.springframework.context.*;
import org.springframework.context.support.*;
import java.io.*;
public class   scheduledemo
{
   public static void main(String args[])
```


JPG to PDF Converter For Mac - Unregis

```
    {
    ApplicationContext ctx=new FileSystemXmlApplicationContext("Timer.xml");
    }
}
-----------------------------------------------
==================================================================================
App 51)>>>>>>>>>>>>>>>Struts with Spring Using Approach No:1<<<<<<<<<<<<<<<<<<<<<<<
==================================================================================
-----------------------------------index1.jsp-----------------------------------
<%@ taglib prefix="html" uri="struts-html" %>
<html>
<body bgcolor=#ffffcc text=green>
<h1><center>struts framewrok</h1><hr><br><h3>
<html:form action="/demo">
  enter user name :<html:text property="uname" /> <br><br>
  <html:submit value="wish" />
</html:form>
</body>
</html>
-----------------------------------web.xml-----------------------------------
<!DOCTYPE web-app
 PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">

<web-app>
 <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>
 <welcome-file-list>
     <welcome-file>index1.jsp</welcome-file>
 </welcome-file-list>
 <taglib>
     <taglib-uri>struts-html</taglib-uri>
     <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
 </taglib>
</web-app>
-----------------------------------struts-config.xml-----------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
1.1//EN"
                "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
<struts-config>
 <form-beans>
  <form-bean name="demofrm" type="demopack.DemoForm" />
 </form-beans>
 <action-mappings>
  <action path="/demo" name="demofrm" type="demopack.DemoAction">
     <forward name="success" path="/result.jsp" />
  </action>
 </action-mappings>
 <plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">
```

form page (View layer)
ref. 4923
action path of StrutsAction class (ref. 4986)
DD file
over ActionServlet configuration
welcome-file configuration.
struts tag library configuration
Struts configuration file (Controller Layer)
over form bean configuration
struts action class configuration
applied plugin for struts application to get access to spring container

JPG to PDF Converter For Mac - Unregis

```
</plug-in>
</struts-config>
----------------------------DemoForm.java----------------------------
package demopack;
import org.apache.struts.action.*;
public class DemoForm  extends ActionForm
{
  private  String uname;

   public void setUname(String s){uname=s;}
   public String getUname(){return uname;}
}
----------------------------DemoAction.java----------------------------
package demopack;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import org.apache.struts.action.*;
import org.springframework.web.struts.*;
import org.springframework.web.context.*;

public class DemoAction extends ActionSupport
{
  public ActionForward execute(ActionMapping mapping,ActionForm form,HttpServletRequest
  request,HttpServletResponse response)
  throws ServletException,IOException
  {
  System.out.println("In execute method of Action class");
  DemoForm   f1=(DemoForm)form;
  String  uname=f1.getUname();

 WebApplicationContext context=getWebApplicationContext();
  ModelInter m1=(ModelInter)context.getBean("d1");
  String msg=m1.getWish()+" "+uname;
 request.setAttribute("message",msg);

  return mapping.findForward("success");
  }
}
----------------------------result.jsp----------------------------
<%@  page language="java" %>
<%@ taglib  prefix="html"  uri="struts-html"  %>
<html>
<body bgcolor=#ffffcc text=green>
<h1><center>struts framewrok</h1><hr><br><h3>
<%=request.getAttribute("message")  %>
<br><br>
<html:link  href="index1.jsp" >home</html:link>
</body>
</html>
----------------------------ModelInter.java----------------------------
package demopack;
public interface ModelInter
{
     public  String getWish();
}
----------------------------ModelImpl.java----------------------------
package demopack;
import java.util.*;
public class ModelImpl implements ModelInter
{
```

FormBean class (controller layer)

Action class (controller layer)

servlet api

struts api

spring api

mandatory to make StrutsAction class as spring aware class

gets springBean

typecasting

reads form data from formBean class

gives webapplication Context Container activated by contextloader plugin

Calling Business method (ref: 5020)

ref: 5020

Result page (view layer)

reads the result from request attribute (ref: 5020)

Keeping the result in request attri to use in the result page

spring Interface

declaration of business method

spring bean class

JPG to PDF Converter For Mac - Unregis

StrutsSpringApp1
- WEB-INF
  - classes
  - lib
    - *.jar (10+172)
  - web.xml
  - struts-config.xml
  - struts-html.tld
  - action-servlet.xml (size 4957)
- index.jsp
- result.jsp

```
public String getWish()
{
System.out.println("getWish() ModelImpl");
   Calendar c1=Calendar.getInstance();
    int h=c1.get(Calendar.HOUR_OF_DAY);

    if(h  < 12)
    return "Good Morning";
    else if( h < 17)
    return "Good Afternoon";
    else
    return "Good Evening";
 }||
}
```

logical name of servlet.xml

spring configuration file.

classes
- *.java(4)
- demopack
  - DemoForm.class
  - DemoAction.class
  - ModelInterface
  - ModelImpl.class

```
----------------------------action-servlet.xml----------------------------
<?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="d1" class="demopack.ModelImpl"/>
</beans>
```

Spring bean class configuration

```
==================================================================
App  52)>>>>>>>>>>>>>>>>>struts-spring (Approach No:2)<<<<<<<<<<<<<<<<<<<<<<<<<<<
==================================================================
----------------------------------index1.jsp----------------------------------
<%@ taglib  prefix="html"  uri="struts-html"  %>
<html>
<body bgcolor=#ffffcc text=green>
<h1><center>struts framewrok</h1><hr><br><h3>
<html:form action="/demo">
  enter user name :<html:text  property="uname"  /> <br><br>
  <html:submit  value="wish" />
</html:form>
</body>
</html>
----------------------------------web.xml----------------------------------
<!DOCTYPE web-app
 PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">
<web-app>
 <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>

 <welcome-file-list>
     <welcome-file>index1.jsp</welcome-file>
 </welcome-file-list>
 <taglib>
     <taglib-uri>struts-html</taglib-uri>
```

for files in class path:
- servlet-api.jar
- spring.jar
- spring-webmvc-struts.jar
- struts-core-1.3.8.jar

for files in WEB-INF/lib folder
- 2 spring.jar files
- 10 struts jar files
- spring-web-mvc-struts.jar

request url:
http://localhost:1020/StrutsSpringApp1/index.jsp

JPG to PDF Converter For Mac - Unregis

```
      <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
  </taglib>
</web-app>
-----------------------------------struts-config.xml-----------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
1.1//EN"
                    "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">

<struts-config>
 <form-beans>
  <form-bean  name="demofrm"  type="demopack.DemoForm" />
 </form-beans>
 <action-mappings>
  <action  path="/demo"  name="demofrm"
  type="org.springframework.web.struts.DelegatingActionProxy" >
    <forward  name="success"  path="/result.jsp" />
  </action>
 </action-mappings>
  <plug-in className="org.springframework.web.struts.ContextLoaderPlugIn"></plug-in>
</struts-config>
-------------------action-servlet.xml--------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
 <bean  name="/demo"  class="demopack.DemoAction" >
   <property  name="model"><ref bean="d1" /></property>
 </bean>
 <bean  id="d1" class="demopack.ModelImpl" >
 </bean>
</beans>
---------------------------DemoForm.java-----------------------------------
package demopack;
import org.apache.struts.action.*;
public class DemoForm  extends ActionForm
{
   private  String uname;
   public void setUname(String s)
    {
       uname=s;
    }
    public String getUname()
    {
       return uname;
    }
}
---------------------------DemoAction.java-----------------------------------
package demopack;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import org.apache.struts.action.*;
public class DemoAction  extends Action
{
    private   ModelInter  model;

    public void setModel(ModelInter i1){ model=i1;}

    public ActionForward execute(ActionMapping mapping,ActionForm form,
                    HttpServletRequest request,HttpServletResponse response)
                    throws ServletException,IOException
```

this action path must match with bean name of struts action class (ref 5135)

dummy (proxy) Action class that passes the request to the real Action class

struts Action class as spring bean having logic to inject other spring bean obj class

ref 5138

ref 5164

our spring bean

property to hold spring bean class object.

methods supporting setter injection

JPG to PDF Converter For Mac - Unregis

```
{
DemoForm  f1=(DemoForm)form;
String  uname=f1.getUname();

String msg=model.getWish()+" "+ uname;

 request.setAttribute("message",msg);
 return mapping.findForward("success");
 }
}
```

calling business method.

Deployment Directory structure

Same as App(51) ✓

```
----------------------------ModelInter.java----------------------------
package demopack;
public interface ModelInter
{
    public  String getWish();
}
----------------------------------------ModelImpl.java----------------------------
package demopack;
import java.util.*;
public class ModelImpl implements ModelInter
{
public  String getWish()
 {
   Calendar c1=Calendar.getInstance();
   int h=c1.get(Calendar.HOUR_OF_DAY);
   if(h  < 12)
      return "Good Morning";
   else if( h < 17)
      return "Good Afternoon";
   else
      return "Good Evening";
 }
}

-------------------------------result.jsp----------------------------
%@  page language="java" %>
<%@ taglib  prefix="html"  uri="struts-html"  %>
<html>
<body bgcolor=#ffffcc text=green>
<h1><center>struts framewrok</h1><hr><br><h3>
 <%= request.getAttribute("message")  %>
<br><br>
<html:link  href="index1.jsp" >home</html:link>
</body>
</html>
--------------------------------------------------------------------------------
==========================================>>>
```

```
<html:radio  property="role" value="Administrator">Administrator</html:radio>
<td><html:radio  property="role" value="User">User</html:radio>
</tr>
<tr>
<td><html:radio  property="role" value="Salesman">Salesman</html:radio>
<td>  <html:radio  property="role" value="Manager">Manager</html:radio>
</tr>
</table>
  <html:submit  value="save" />
</html:form>
</body>
</html>
--------------------------------result.jsp-----------------------------
%@  page language="java" %>
<%@ taglib  prefix="html"  uri="struts-html"  %>
<html>
<body bgcolor=#ffffcc text=green>
<h1><center>welcome user</h1><hr><br><h3>
</body>
</html>
-------------------------------------------web.xml-----------------------
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_3.dtd">
<web-app>
 <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>

   <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
   </welcome-file-list>

   <taglib>
    <taglib-uri>struts-html</taglib-uri>
    <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
   </taglib>
</web-app>
-----------------------------struts-config.xml-----------------------------
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
1.1//EN"
                    "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
<struts-config>
  <form-beans>
    <form-bean  name="userform"   type="demo.UserForm" />
  </form-beans>
  <action-mappings>
    <action  path="/user"  name="userform"
    type="org.springframework.web.struts.DelegatingActionProxy">
```

JPG to PDF Converter For Mac - Unregis

```
    <forward name="success" path="/result.jsp" />
    <forward name="failure" path="/index.jsp" />
  </action>
</action-mappings>
<plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">
  </plug-in>
</struts-config>
```

----------UserForm.java----------

```
package demo;
import org.apache.struts.action.*;
public class UserForm extends ActionForm
{
 private String uname,role;

 public void setUname(String s){uname=s;}
 public String getUname(){return uname;}
 public void setRole(String s){ role=s; }
 public String getRole(){ return role; }
}
```

----------UserAction.java----------

```
package demo;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import org.apache.struts.action.*;
public class UserAction extends Action
{  private  ModelInter model;

public void setModel(ModelInter i1) { model=i1; }
 public ActionForward execute(ActionMapping mapping,ActionForm form,
                    HttpServletRequest request,HttpServletResponse response)
                    throws ServletException,IOException
 {
 UserForm  f1=(UserForm)form;
 String uname=f1.getUname();
 String role=f1.getRole();

  if(model.insertInfo(uname,role))
   return mapping.findForward("success");
   else
    return mapping.findForward("failure");
  }
}
```

----------ModelInter.java----------

```
package demo;
public interface ModelInter
{
     public boolean insertInfo(String uname,String role);
}
```

----------ModelImpl.java----------

```
package demo;
import java.util.*;
import org.hibernate.*;
public class ModelImpl implements ModelInter
{
 private SessionFactory sesfact;

public void setSesfact(SessionFactory sf)
{
     sesfact=sf;
 }
```


JPG to PDF Converter For Mac - Unregis

```
public  boolean insertInfo(String uname,String role)
{       Session ses=sesfact.openSession();
        Transaction tx=ses.beginTransaction();

        boolean flag=false;
        try{
          User u1=new User();
            u1.setUname(uname);
        u1.setRole(role);
        ses.save(u1);
        tx.commit();
        flag=true;
        }catch(Exception e)
        {       tx.rollback();
        System.out.println(e);
 }
        ses.close();
        return flag;
}
}
--------------------------------action-servlet.xml--------------------------
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
<beans>

 <bean  name="/user"   class="demo.UserAction" >
   <property  name="model"><ref bean="d1" /></property>
 </bean>

 <bean   id="d1" class="demo.ModelImpl" >
  <property  name="sesfact"><ref bean="mySessionFactory" /></property>
 </bean>
 <bean id="myDataSource"
 class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName"><value>oracle.jdbc.driver.OracleDriver</value></property>
    <property name="url">value>jdbc:oracle:thin:@localhost:1521:orcl</value></property>
    <property name="username"><value>scott</value></property>
    <property name="password"><value>tiger</value></property>
  </bean>
  <bean id="mySessionFactory"
  class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="myDataSource"/>
     <property name="mappingResources">
      <list>
        <value>User.hbm.xml</value>
      </list>
    </property>
    <property name="hibernateProperties">
      <props>
        <prop key="hibernate.dialect">org.hibernate.dialect.Oracle9Dialect</prop>
     </props>
    </property>
  </bean>
</beans>
--------------------------------User.java--------------------------------
package demo;
public class User
{
   private int uid;
   private String uname,role;
```

JPG to PDF Converter For Mac - Unregis

```
public void setUid(int n){uid=n;}
public int getUid(){ return uid; }
public void setUname(String s){ uname=s; }
public String getUname(){ return uname; }
public void setRole(String r){ role=r; }
public String getRole(){  return role; }
}
```

In spring command class property names and form component names need not be matched

----------user.hbm.xml----------

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
 <class    name="demo.User"   table="users" >
  <id  name="uid"  column="userid" >
     <generator  class="increment" />
  </id>
  <property   name="uname" />
  <property   name="role" />
 </class>
</hibernate-mapping>
```

==========spring MVC==========

App 54)>>>>>>>>>>>>>>>>Application on spring MVC<<

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">
<servlet>
  <servlet-name>productaction</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>productaction</servlet-name>
  <url-pattern>*.htm</url-pattern>
</servlet-mapping>
<taglib>
  <taglib-uri>spring-tld</taglib-uri>
  <taglib-location>/WEB-INF/spring.tld</taglib-location>
</taglib>
</web-app>
----------productaction-servlet.xml----------
<?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="simpleUrlMappings"
  class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping" >
    <property name="mappings">
      <props>
       <prop key="/newpro.htm">newproController</prop>
      </props>
    </property>
  </bean>

  <bean id="newproController" class="NewProductController" >
    <property name="sessionForm"> <value>true</value></property>
    <property name="commandName"> <value>p</value></property>
    <property name="commandClass"> <value>Product</value></property>
    <property name="formView"> <value>newproduct</value></property>
    <property name="successView"> <value>showproduct</value></property>
  </bean>

  <bean id="viewResolver" class =
  "org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property
    name="viewClass"><value>org.springframework.web.servlet.view.JstlView</value></property>
    <property name ="prefix"><value>/</value></property>
    <property name ="suffix"><value>.jsp</value></property>
  </bean>
</beans>
----------Product.java----------
public class Product
{
   private int pid;
   private String pname;
   private double price;
   public void setPid(int n){ pid=n; }
   public int getPid(){ return pid; }
   public void setPname(String s){ pname=s; }
   public String getPname(){ return pname; }
   public void setPrice(double p){ price=p; }
   public double getPrice(){ return price; }
```

extension match url pattern to make Dispatcher servlet as front controller

spring supplied jsp tag lib configuration

spring cfg file

logical name of Dispatcher Servlet configuration

to link the form with commandcontroller class

bean id of commandcontroller class (ref 5502)

command controller class

ref 5437

command class name

to use jsp pages as view resolver

makes the view logical name command class jsp programs getting '/' as prefix .jsp as suffix [refer 5506, 5507]

*1 → formview specifies the form page name, the result page name. But here only file names are specified because '/' as prefix, .jsp as suffix, will be getting automatically through viewResolver configuration (5510 to 5514)

JPG to PDF Converter For Mac - Unregis

command controller class

```
}

----------------------------NewProductController.java----------------------------
import org.springframework.web.servlet.mvc.*;
import org.springframework.web.servlet.*;
import   org.springframework.validation.*;
import javax.servlet.http.*;
public class   NewProductController  extends SimpleFormController
{
 public ModelAndView  onSubmit(HttpServletRequest request,HttpServletResponse response,
                        Object command,BindException e) throws Exception
 {
  Product p=(Product)command;
  String str="Product id :"+p.getPid()+" <br> "
      +"product name :"+p.getPname()+"<br> "
     +"Price :"+p.getPrice();

  if(p.getPrice()>=10000)
     request.setAttribute("msg","costly");
   else
     request.setAttribute("msg","cheap");


  System.out.println(str);
  request.setAttribute("productinfo",str);
  return new ModelAndView(getSuccessView());
  //return new ModelAndView("showproduct");
   }
}
```

logic to render single string having the entire data of command class object.

returns the model and view class obj pointing to the showproduct.jsp

select page

```
----------------------------showproduct.jsp----------------------------
<html>
<body bgcolor=#ffffcc text=green>
<h1><center>new product information</h1><hr><br><h3>
 <%=request.getAttribute("productinfo") %> <br>
 Item is < <%=request.getAttribute("msg") %>
</body>
</html>
```

ref line no 5558

ref. 5546 or 5548

```
-----------------------------------------------------------------------------------------
=========================================================================================
App 55 )>>>>>>>>>>>>>>>Application on SpringMVC having form validations <<<<<<<<<<<<<<
=========================================================================================
==========================index.jsp==========================================================
<jsp:forward   page="/newpro.htm"  />
==========================newproduct.jsp=====================================================
<%@ taglib  prefix="spring"  uri="spring-tld"  %>
<%@ taglib  prefix="c"  uri="core" %>
<html>
<body bgcolor=#ffffcc text=green>
<h1><center>new product information</h1><hr><br><h3>
<form method="post" >

enter product id :
<spring.bind  path="p.pid" >
 <input type=text name=pid />
</spring.bind>
<br><br>

enter product name :
<spring.bind  path="p.pname" >
 <input type=text name=pname />
</spring.bind>
<br><br>
```


JPG to PDF Converter For Mac - Unregis

```
5589
5590
5591  enter product price :
5592  <spring.bind  path="p.price" >
5593    <input type=text name=price />
5594  </spring.bind>
5595  <br><br>
5596   <input type=submit value=send />
5597
5598  </form>
5599
5600   <spring:hasBindErrors name="p">
5601     <ul>
5602       <c:forEach var="e" items="${errors.allErrors}">
5603         <li>
5604          <spring:message text="${e.defaultMessage}"/>
5605         </li>
5606       </c:forEach>
5607     </ul>
5608   </spring:hasBindErrors>
5609  </body>
5610  </html>
5611  ------------------------------web.xml----------------------------------------
5612  <?xml version="1.0" encoding="ISO-8859-1"?>
5613  <web-app xmlns="http://java.sun.com/xml/ns/j2ee"
5614     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5615     xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
5616     http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
5617     version="2.4">
5618
5619      <servlet>
5620        <servlet-name>productaction</servlet-name>
5621        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
5622        <load-on-startup>1</load-on-startup>
5623     </servlet>
5624
5625     <servlet-mapping>
5626        <servlet-name>productaction</servlet-name>
5627        <url-pattern>*.htm</url-pattern>
5628     </servlet-mapping>
5629     <taglib>
5630        <taglib-uri>spring-tld</taglib-uri>
5631        <taglib-location>/WEB-INF/spring.tld</taglib-location>
5632      </taglib>
5633     <taglib>
5634        <taglib-uri>core</taglib-uri>
5635        <taglib-location>/WEB-INF/c.tld</taglib-location>
5636     </taglib>
5637
5638  </web-app>
5639
5640  ------------------------------productaction-servlet.xml----------------------------
5641  <?xml version="1.0" encoding="UTF-8"?>
5642  <!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
5643  "http://www.springframework.org/dtd/spring-beans.dtd">
5644  <beans>
5645    <bean  id="simpleUrlMapping"
       class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping" >
5646      <property  name="mappings">
5647        <props>
5648          <prop  key="/newpro.htm">newpro</prop>
```

refer line no: 5656

represents all form validation errors

displays each form validation error message

logic to display all form validation errors

jstl core tag library

JPG to PDF Converter For Mac - Unregis

```
    </props>
  </property>
 </bean>

 <bean  id="newpro"  class="NewProductController" >
   <property name="validator"><bean class="ProductValidator"/></property>
   <property name="sessionForm" > <value>true</value></property>
   <property name="commandName" > <value>p</value></property>
   <property name="commandClass" > <value>Product</value></property>
    <property name="formView" > <value>newproduct</value></property>
   <property name="successView" > <value>showproduct</value></property>
 </bean>

 <bean id="viewResolver" class =
 "org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property
  name="viewClass"><value>org.springframework.web.servlet.view.JstlView</value></property>
   <property name ="prefix"><value>/</value></property>
   <property name ="suffix"><value>.jsp</value></property>
  </bean>
 </beans>
 ----------------------------------Product.java----------------------------------
 public class Product
 {
   private  int pid;
   private String pname;
   private double price;

   public void setPid(int n){ pid=n; }
   public int getPid(){ return pid; }
   public void setPname(String s){ pname=s; }
   public String getPname(){ return pname; }
   public void setPrice(double p){ price=p; }
   public double getPrice(){ return price; }
 }
 ----------------------------------ProductValidator.java----------------------------------
 import org.springframework.validation.*;

 public class  ProductValidator implements Validator
 {

 public boolean supports(Class c)  //verifies weather appropriate cmd class is passed passed
                                   //as the argument validate () or not.
 {
  return c.equals(Product.class);
 }

 public void validate(Object cmd,Errors errs)
 {
  System.out.println("validate of ProductValidator");
  Product p=(Product)cmd;
  ValidationUtils.rejectIfEmptyOrWhitespace(errs, "pid", "", "id is requied");
  ValidationUtils.rejectIfEmptyOrWhitespace(errs, "pname", "", "name is requied");
  ValidationUtils.rejectIfEmptyOrWhitespace(errs, "price", "", "price is requied");

  String pna=p.getPname();
  if( pna.trim().length() <5)
  errs.rejectValue("pname","","name must have at least 5 characters");
  }
 }
```

mandatory

checks weather this class is pointing to correct command class or not

method having form validation logic

default error message

↳ argument to specify the key of properties file

↳ Keep this class in WEB-INF/classes folder

JPG to PDF Converter For Mac - Unregis

```
----------------------------------NewProductController.java----------------------------------
import org.springframework.web.servlet.mvc.*;
import org.springframework.web.servlet.*;
import   org.springframework.validation.*;
import javax.servlet.http.*;
public class   NewProductController  extends SimpleFormController
{
  public ModelAndView  onSubmit(HttpServletRequest request,HttpServletResponse response,
                Object command,
                BindException e) throws Exception
  {
   Product p=(Product)command;
   String str="Product id :"+p.getPid()+" <br>  "
        +"product name :"+p.getPname()+"<br>  "
        +"Price :"+p.getPrice();
        System.out.println(str);

     request.setAttribute("productinfo",str);
    return new ModelAndView(getSuccessView());
 //   return new ModelAndView("showproduct");
  }
}
-----------------------------showproduct.jsp-----------------------------
<%@ page language="java" %>
<html>
<body bgcolor=#ffffcc text=green>
<h1><center>new product information</h1><hr><br><h3>
 <%=request.getAttribute("productinfo") %>
 <br>
</body>
</html>
-----------------------------------
==========================================================================
App 56)>>>>>>>>>>>>>>>>Example on spring mvc with annotations<<<<<<<<<<<<<<<<<<<<
==========================================================================
-----------------------index.jsp-------------------

<jsp:forward page="pro.htm"/>
----------------------userForm.jsp-----------------------

<form method="POST" action="pro.htm">
 User Name :<input type=text  name="name" /><br>
 Password :<input type=password name=password /><br>
 <input type="submit">

</form>
--------------web.xml-----------------
<web-app>
  <servlet>
      <servlet-name>dispatcher</servlet-name>
      <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  </servlet>
   <servlet-mapping>
      <servlet-name>dispatcher</servlet-name>
      <url-pattern>*.htm</url-pattern>
   </servlet-mapping>
</web-app>
-----------------dispatcher-servlet.xml-----------
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

JPG to PDF Converter For Mac - Unregis

```
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/context"
                http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">

 <context:component-scan base-package="p1" />
</beans>
--------------------------User.java----------------
package p1;

public class User {

private String name;
private String password;

 //write getXxx() and setXxx(-) methods

public String getName()
{
     System.out.println("getName()");
     return name;
}
public void setName(String name)
{
     System.out.println("setName(-)");
     this.name = name;
}
public String getPassword()
{
 System.out.println("getPassword()");
 return password;
}
public void setPassword(String password)
{
 System.out.println("setPassword(-)");
     this.password = password;
}
}
--------------------UserController.java----------------
package p1;

import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.*;
//import java.util.Date;

@Controller
@RequestMapping("/pro.htm")
//@SessionAttributes("msg")

public class UserController
{

@RequestMapping(method = RequestMethod.GET)
public String showUserForm(ModelMap model)
{
     System.out.println("showUserForm(-) executed");
     User u1 = new User();
     model.addAttribute(u1);
```

JPG to PDF Converter For Mac - Unregis

```
        return "userForm.jsp";
    }

@RequestMapping(method = RequestMethod.POST)
public String mySubmit(@ModelAttribute("user") User user)
{
        System.out.println("mySubmit(-)");
         //read form data from Command class obj
        String uname=user.getName();
        String pass=user.getPassword();
        //write b.logic
        if(uname.equals("durga")&& pass.equals("soft"))
        {
         return "userSuccess.jsp";
        }
        else
        {
        return "userFailure.jsp";
        }
    }//mySubmit(-)

}//class
----------------------userSuccess.jsp--------------
ValidCredentials
--------------------------------userFailure.jsp----------------
InValidCredentials
-----------------------------------------------------------
=================================================================================
App 57)>>>>>>>>>>>>>>>>>Example on spring mvc MultiActionController<<<<<<<<<<<<<<<
=================================================================================
---------------------------user.jsp-------------------
<html>
  <body>
    <center>
    <form method="POST" action="abc.cpp">
      Sno :<input type=text name="no"><br>
      SName:<input type=text name="name" /><br>
      Sadd:<input type=text name="add" /><br>
      Email:<input type=text name="email"/><br>

      <input type="submit" name="btn" value="insert"/>
      <input type="submit" name="btn" value="update"/>
      <input type="submit" name="btn" value="delete"/>
      <input type="submit" name="btn1" value="abc"/>
    </form>
   </center>
  </body>
 </html>
 ------------------------------------web.xml------------------------------------

 <web-app>
    <servlet>
       <servlet-name>disp</servlet-name>
       <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    </servlet>

    <servlet-mapping>
       <servlet-name>disp</servlet-name>
       <url-pattern>*.cpp</url-pattern>
    </servlet-mapping>
```

JPG to PDF Converter For Mac - Unregis

```
<welcome-file-list>
  <welcome-file>user.jsp</welcome-file>
</welcome-file-list>

</web-app>
```

----------disp-servlet.xml----------

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<bean name="daobean" class="p1.UserDAO"/>

<bean id="mnr"
class="org.springframework.web.servlet.mvc.multiaction.ParameterMethodNameResolver">
 <property name="paramName" value="btn"/>
 <property name="defaultMethodName" value="requestNotFound"/>
</bean>
<bean name="/*.cpp" class="p1.UserController">
 <property name="udao" ref="daobean"/>
 <property name="methodNameResolver" ref="mnr"/>
</bean>

</beans>
```

----------UserCommand.java----------

```
package p1;
public class UserCommand
{
int no;
String name;
String add;
String email;

public UserCommand()
{
 System.out.println("UserCommand()");
 }

public void setNo(int x){  no=x;}
public void setName(String s1){ name=s1;}
public void setAdd(String add){ this.add=add;}
public void setEmail(String mail){   email=mail;}

 public int getNo(){return no;}
 public String getName(){return name;}
 public String getAdd(){       return add;}
 public String getEmail(){return email;}
}
```

----------UserController.java----------

```
package p1;

import java.io.IOException;
import javax.servlet.http.*;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.multiaction.MultiActionController;

public class UserController extends MultiActionController
{
```

JPG to PDF Converter For Mac - Unregis

```
UserDAO udao;

public UserController()
{
 System.out.println("my multi controller is executed");
}

public void setUdao(UserDAO udao)
{this.udao=udao;}


public ModelAndView insert(HttpServletRequest request,HttpServletResponse
response,UserCommand uc)
{
   System.out.println("insert() called");
   int i=0;
   try{
   i=udao.add(uc);
  }catch(Exception e){}

  return new ModelAndView("success.jsp","result",i+" Record Inserted Successfully");
}

public ModelAndView update(HttpServletRequest request,HttpServletResponse
response,UserCommand uc)
{
 System.out.println("update() called");
  int i=0;

   try{
        i=udao.edit(uc);
    }catch(Exception e){}

  return new ModelAndView("success.jsp","result",i+"Record updated Successfully");
}

public ModelAndView delete(HttpServletRequest request,HttpServletResponse
response,UserCommand uc)
{

System.out.println("delete() called");
int i=0;
try{
        i=udao.remove(uc);
  }catch(Exception e){}

  return new ModelAndView("success.jsp","result",i+"Record Deleted Successful");
}

public ModelAndView requestNotFound(HttpServletRequest request,HttpServletResponse
response,UserCommand uc){
        return new ModelAndView("success.jsp","result","Invalid Request");
    }
}
----------------------------UserDAO.java----------------------------
package p1;
import java.sql.*;
/*
create table student1
(
```

JPG to PDF Converter For Mac - Unregis

```
6007  sno number(5),
6008   sname varchar2(20),
6009   sadd varchar2(20),
6010   email varchar2(20)
6011 );
6012
6013 */
6014 public class UserDAO
6015 {
6016  Connection con;
6017  PreparedStatement pst;
6018  ResultSet rs;
6019
6020 public Connection getConn()
6021 {
6022  try
6023  {
6024   Class.forName("oracle.jdbc.driver.OracleDriver");
6025   con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
6026   return con;
6027  }catch(Exception e){System.out.println("Exception occured"+e);return null;}
6028 }
6029 public void closeConn()
6030 {
6031  try{
6032   con.close();
6033   }catch(Exception e){System.out.println("Error while closing connection");}
6034  }
6035
6036
6037 public int add(UserCommand uc)throws Exception
6038 {
6039  con=getConn();
6040  pst=con.prepareStatement("insert into student1 values(?,?,?,?)");
6041
6042  pst.setInt(1,uc.getNo());
6043  pst.setString(2,uc.getName());
6044  pst.setString(3,uc.getAdd());
6045  pst.setString(4,uc.getEmail());
6046
6047  int i=pst.executeUpdate();
6048
6049   return i;
6050  }
6051
6052 public int edit(UserCommand uc)throws Exception
6053  {
6054  con=getConn();
6055  pst=con.prepareStatement("update student1 set sname=?,sadd=?,email=? where sno=? ");
6056  pst.setString(1,uc.getName());
6057  pst.setString(2,uc.getAdd());
6058  pst.setString(3,uc.getEmail());
6059  pst.setInt(4,uc.getNo());
6060
6061  int i=pst.executeUpdate();
6062   return i;
6063  }
6064
6065 public int remove(UserCommand uc)throws Exception
6066  {
6067   con=getConn();
```


JPG to PDF Converter For Mac - Unregis

```
pst=con.prepareStatement("delete from student1 where sno=?");
pst.setInt(1,uc.getNo());
int i=pst.executeUpdate();
return i;
}


}
----------------success.jsp-------------------
<%= request.getAttribute("result")
%>
==========================================
App 88)Example on spring mvc to render pdf/excel type response
==========================================
---------------------input.html-------------
<form action="spring.do">
    <input type="submit">
  </form>
-------------------------------web.xml--------------------------
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
   version="2.4">

    <servlet>
      <servlet-name>productaction</servlet-name>
      <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
      <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
      <servlet-name>productaction</servlet-name>
      <url-pattern>*.do</url-pattern>
    </servlet-mapping>


</web-app>
------------------------productaction-servlet.xml------------------
<?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="url" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
   <property name="mappings">
    <props>
      <prop key="spring.do">controller</prop>
    </props>
   </property>
  </bean>
  <bean id="controller" class="TestController"/>
  <bean id="bnvr" class="org.springframework.web.servlet.view.BeanNameViewResolver"/>
 <!-- <bean id="viewbean" class="myexcelview"/> -->
 <bean id="viewbean" class="mypdfview"/>

</beans>
-----------------------------TestController.java-----------------
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

command controller class Config

spring beans acting as view Layer Resolver

always to use spring beans as views based on the bean name returned by command controller class as part of ModelandView class object

JPG to PDF Converter For Mac - Unregis

```
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;
import java.util.*;

public class TestController implements Controller
{

public ModelAndView handleRequest(HttpServletRequest req,
        HttpServletResponse res) throws Exception {
    List l = new ArrayList();
    l.add("abcd");
    l.add("xyz");
    l.add("test");
    return new ModelAndView("viewbean","result",l);
}
}
```

→ the beanid of springbean that acts as view layer resource (cot 6/2/ or cot 6/22)

↳ requestattribute.

----------myexcelview.java----------

↳ spring bean as view layer resource to generate output in the form of MS-Excel document

```
import java.util.*;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.poi.hssf.usermodel.*;
import org.springframework.web.servlet.view.document.*;

public class myexcelview extends AbstractExcelView {

protected void buildExcelDocument(Map m, HSSFWorkbook wb,
        HttpServletRequest req, HttpServletResponse res) throws Exception {
    HSSFSheet sheet=wb.createSheet("Names");
    ArrayList l =(ArrayList)m.get("result");
    getCell(sheet,0,0).setCellValue(new HSSFRichTextString("User Names"));
    getCell(sheet,1,0).setCellValue(new HSSFRichTextString(l.get(0)+" "));
    getCell(sheet,2,0).setCellValue(new HSSFRichTextString(l.get(1)+" "));
    getCell(sheet,3,0).setCellValue(new HSSFRichTextString(l.get(2)+" "));
    }
}
```

→ This internally uses JExcelApi to render the excel document ✓

→ holds attributes given by command controller class ✓

→ reads the requestattribute value ✓

----------mypdfview.java----------

→ Constructing the xl sheet based on the data of arraylist object ✓

```
import java.util.*;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.view.document.AbstractPdfView;
import com.lowagie.text.*;
import com.lowagie.text.pdf.PdfWriter;

public class mypdfview extends AbstractPdfView {

protected void buildPdfDocument(Map arg0, Document arg1, PdfWriter arg2,
        HttpServletRequest arg3, HttpServletResponse arg4) throws Exception {
    Object o = arg0.get("result");
    ArrayList l =(ArrayList)o;
    Paragraph p = new Paragraph("User details");
    p.setAlignment("center");
    arg1.add(p);(*)

    Table t = new Table(1);
    t.addCell(l.get(0)+" ");
    t.addCell(l.get(1)+" ");
    t.addCell(l.get(2)+" ");
    arg1.add(t);
```

→ spring bean as view layer resource to generate pdf document as response by using bruno lowagie's iText package [API]

→ reading attribute given by command controller class ✓

→ creates one paragraph in pdf document ✓

→ preparing table having content

↳ adds table to pdf document ✓

(*) → adds paragraph to pdf document

JPG to PDF Converter For Mac - Unregis

springpdfApp

- → WEB-INF
  - → classes
  - → lib
    - → *.jar
  - → web.xml
  - → productaction-servlet.xml
- → input.html

```
6190
6191  }
6192  }
6193
```

order classes folder

classes
- TestController.class
- myPdfView.class
- → myExcelView.class

→ jar files in class path:-

spring.jar
spring-webmvc.jar
iText-2.0.7.jar
pdf-3.2-Final-20081019.jar → some no

jar files in WEB-INF/lib

→ The beside jar files
commons-logging.jar

→ request url: https://localhost:8080/springpdfApp/input.html

JPG to PDF Converter For Mac - Unregis

Object Relational Mapping (ORM) data access

The Spring Framework provides integration with Hibernate, JDO, Oracle TopLink, iBATIS SQL Maps and JPA: in terms of resource management, DAO implementation support, and transaction strategies. All of these support packages for O/R (Object Relational) mappers comply with Spring's generic transaction and DAO exception hierarchies.

There are usually two integration styles: either using Spring's DAO 'templates' or coding DAOs against plain Hibernate/JDO/TopLink/etc APIs. In both cases, DAOs can be configured through Dependency Injection and participate in Spring's resource and transaction management.

Some of the benefits of using the Spring Framework to create your ORM DAOs include:

- Ease of testing. Spring's IoC approach makes it easy to swap the implementations and config locations of Hibernate SessionFactory instances, JDBC DataSource instances, transaction managers, and mappes object implementations (if needed). This makes it much easier to isolate and test each piece of persistence-related code in isolation.
- Common data access exceptions. Spring can wrap exceptions from your O/R mapping tool of choice, converting them from proprietary (potentially checked) exceptions to a common runtime DataAccessException hierarchy. This allows you to handle most persistence exceptions, which are non-recoverable, only in the appropriate layers, without annoying boilerplate catches/throws, and exception declarations. You can still trap and handle exceptions anywhere you need to. Remember that JDBC exceptions (including DB specific dialects) are also converted to the same hierarchy, meaning that you can perform some operations with JDBC within a consistent programming model.
- General resource management. Spring application contexts can handle the location and configuration of Hibernate SessionFactory instances, JDBC DataSource instances, iBATIS SQL Maps configuration objects, and other related resources. This makes these values easy to manage and change. Spring offers efficient, easy and safe handling of persistence resources. For example: related code using Hibernate generally needs to use the same Hibernate Session for efficiency and proper transaction handling. Spring makes it easy to transparently create and bind a Session to the current thread, either by using an explicit 'template' wrapper class at the Java code level or by exposing a current Session through the Hibernate SessionFactory (for DAOs based on plain Hibernate API). Thus Spring solves many of the issues that repeatedly arise from typical Hibernate usage, for any transaction environment (local or JTA).
- Integrated transaction management. Spring allows you to wrap your O/R mapping code with either a declarative, AOP style method interceptor, or an explicit 'template' wrapper class at the Java code level. In either case, transaction semantics are handled for you, and proper transaction handling (rollback, etc) in case of exceptions is taken care of. As discussed below, you also get the benefit of being able to use and swap various transaction managers, without your Hibernate/JDO related code being affected: for example, between local transactions and JTA, with the same full services (such as declarative transactions) available in both scenarios. As an additional benefit, JDBC-related code can fully integrate transactionally with the code you use to do O/R mapping. This is useful for data access that's not suitable for O/R mapping, such as batch processing or streaming of BLOBs, which still needs to share common transactions with ORM operations.

As of Spring 2.5, Spring requires Hibernate 3.1 or higher. Neither Hibernate 2.1 nor Hibernate 3.0 are supported anymore.

JPG to PDF Converter For Mac - Unregis

DAO definition in a Spring container - HibernateTemplate method

The HibernateTemplate class provides many methods that mirror the methods exposed on the HibernateSession interface, in addition to a number of convenience methods such as the one shown below.

```
<beans>
    <bean id="myProductDao" class="product.ProductDaoImpl">
        <property name="sessionFactory" ref="mySessionFactory"/>
    </bean>
</beans>
public class ProductDaoImpl implements ProductDao {
    private HibernateTemplate hibernateTemplate;
    public void setSessionFactory(SessionFactory sessionFactory) {
        this.hibernateTemplate = new HibernateTemplate(sessionFactory);
    }
    public Collection loadProductsByCategory(String category)
        throws DataAccessException {
        return this.hibernateTemplate.find("from test.Product product
                where product.category=?", category);
    }
}
```

If you need access to the Session to invoke methods that are not exposed on the HibernateTemplate, you can always drop down to a callback-based approach

```
public class ProductDaoImpl implements ProductDao {
    private HibernateTemplate hibernateTemplate;
    public void setSessionFactory(SessionFactory sessionFactory) {
        this.hibernateTemplate = new HibernateTemplate(sessionFactory);
    }
    public Collection loadProductsByCategory(final String category)
            throws DataAccessException {
        return this.hibernateTemplate.execute(new HibernateCallback() {
            public Object doInHibernate(Session session) {
                Criteria criteria = session.createCriteria(Product.class);
                criteria.add(Expression.eq("category", category));
                criteria.setMaxResults(6);
                return criteria.list();
```

JPG to PDF Converter For Mac - Unregis

JPG to PDF Converter For Mac - Unregis


```
            }
        };
    }
}
```

A callback implementation effectively can be used for any Hibernate data access. HibernateTemplate will ensure that Session instances are properly opened and closed, and automatically participate in transactions. The template instances are thread-safe and reusable, they can thus be kept as instance variables of the surrounding class. For simple single step actions like a single find, load, saveOrUpdate, or delete call, HibernateTemplate offers alternative convenience methods that can replace such one line callback implementations. Furthermore, Spring provides a convenient HibernateDaoSupport base class that provides a setSessionFactory(..) method for receiving a SessionFactory, and getSessionFactory() and getHibernateTemplate()for use by subclasses. In combination, this allows for very simple DAO implementations for typical requirements:

```
public class ProductDaoImpl extends HibernateDaoSupport implements ProductDao {
    public Collection loadProductsByCategory(String category)
        throws DataAccessException {
        return this.getHibernateTemplate().find(
            "from test.Product product where product.category=?", category);
    }
}
```

Implementing Spring-based DAOs without callbacks

As alternative to using Spring's HibernateTemplate to implement DAOs, data access code can also be written in a more traditional fashion, without wrapping the Hibernate access code in a callback, while still respecting and participating in Spring's generic DataAccessException hierarchy. The HibernateDaoSupport base class offers methods to access the current transactional Session and to convert exceptions in such a scenario; similar methods are also available as static helpers on the SessionFactoryUtils class. Note that such code will usually pass 'false' as the value of the getSession(..) methods 'allowCreate' argument, to enforce running within a transaction (which avoids the need to close the returned Session, as its lifecycle is managed by the transaction).

```
public class HibernateProductDao extends HibernateDaoSupport
        implements ProductDao {
    public Collection loadProductsByCategory(String category)
        throws DataAccessException, MyException {
        Session session = getSession(false);
        try {
            Query query = session.createQuery("from test.Product product
                                where product.category=?");
            query.setString(0, category);
```

```
            List result = query.list();
            if (result == null) {
                throw new MyException("No search results.");
            }
            return result;
        }
        catch (HibernateException ex) {
            throw convertHibernateAccessException(ex);
        }
    }
}
```

The advantage of such direct Hibernate access code is that it allows any checked application exception to be thrown within the data access code; contrast this to the HibernateTemplate class which is restricted to throwing only unchecked exceptions within the callback. Note that you can often defer the corresponding checks and the throwing of application exceptions to after the callback, which still allows working with HibernateTemplate. In general, the HibernateTemplate class' convenience methods are simpler and more convenient for many scenarios.

Implementing DAOs based on plain Hibernate 3 API

Hibernate 3 provides a feature called "contextual Sessions", where Hibernate itself manages one current Session per transaction. This is roughly equivalent to Spring's synchronization of one Hibernate Session per transaction. A corresponding DAO implementation looks like as follows based on the plain Hibernate API:

```
public class ProductDaoImpl implements ProductDao {
    private SessionFactory sessionFactory;
    public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }
    public Collection loadProductsByCategory(String category) {
        return this.sessionFactory.getCurrentSession()
            .createQuery("from test.Product product
                        where product.category=?")
            .setParameter(0, category).list();
    }
}
```

This style is very similar to what you will find in the Hibernate reference documentation and examples, except for holding the SessionFactory in an instance variable. We strongly recommend such an instance-based setup over the old-school static HibernateUtil class. (In general, do not keep any resources in static variables unless absolutely necessary.)

JPG to PDF Converter For Mac - Unregis

```
            List result = query.list();
            if (result == null) {
                throw new MyException("No search results.");
            }
            return result;
        }
        catch (HibernateException ex) {
            throw convertHibernateAccessException(ex);
        }
    }
}
```

The advantage of such direct Hibernate access code is that it allows any checked application exception to be thrown within the data access code; contrast this to the HibernateTemplate class which is restricted to throwing only unchecked exceptions within the callback. Note that you can often defer the corresponding checks and the throwing of application exceptions to after the callback, which still allows working with HibernateTemplate. In general, the HibernateTemplate class' convenience methods are simpler and more convenient for many scenarios.

Implementing DAOs based on plain Hibernate 3 API

Hibernate 3 provides a feature called "contextual Sessions", where Hibernate itself manages one current Session per transaction. This is roughly equivalent to Spring's synchronization of one Hibernate Session per transaction. A corresponding DAO implementation looks like as follows, based on the plain Hibernate API:

```
public class ProductDaoImpl implements ProductDao {
    private SessionFactory sessionFactory;
    public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }
    public Collection loadProductsByCategory(String category) {
        return this.sessionFactory.getCurrentSession()
            .createQuery("from test.Product product
                    where product.category=?")
            .setParameter(0, category).list();
    }
}
```

This style is very similar to what you will find in the Hibernate reference documentation and examples, except for holding the SessionFactory in an instance variable. We strongly recommend such an instance-based setup over the old-school static HibernateUtil class. (In general, do not keep any resources in static variables unless absolutely necessary.)

JPG to PDF Converter For Mac - Unregis

The above DAO follows the Dependency Injection pattern: it fits nicely into a Spring IoC container, just like it would if coded against Spring's HibernateTemplate. Of course, such a DAO can also be set up in plain Java (for example, in unit tests): simply instantiate it and call setSessionFactory(..) with the desired factory reference. As a Spring bean definition, it would look as follows:

```
<beans>
    <bean id="myProductDao" class="product.ProductDaoImpl">
        <property name="sessionFactory" ref="mySessionFactory"/>
    </bean>
</beans>
```

The main advantage of this DAO style is that it depends on Hibernate API only; no import of any Spring class is required. This is of course appealing from a non-invasiveness perspective, and will no doubt feel more natural to Hibernate developers.However, the DAO throws plain HibernateException (which is unchecked, so does not have to be declared or caught), which means that callers can only treat exceptions as generally fatal - unless they want to depend on Hibernate's own exception hierarchy. Catching specific causes such as an optimistic locking failure is not possible without tieing the caller to the implementation strategy.

This tradeoff might be acceptable to applications that are strongly Hibernate-based and/or do not need any special exception treatment. Fortunately, Spring's LocalSessionFactoryBean supports Hibernate's SessionFactory.getCurrentSession() method for any Spring transaction strategy, returning the current Spring-managed transactional Session even with HibernateTransactionManager. Of course, the standard behavior of that method remains: returning the current Session associated with the ongoing JTA transaction, if any (no matter whether driven by Spring's JtaTransactionManager, by EJB CMT, or by JTA).

In summary: DAOs can be implemented based on the plain Hibernate 3 API, while still being able to participate in Spring-managed transactions.

JPG to PDF Converter For Mac - Unregis

Spring Remoting

Spring provides integration classes for remoting support using various technologies. The remoting support eases the development of remote-enabled services, implemented by your usual (Spring) POJOs.

Currently, Spring supports the following remoting technologies:

- **Remote Method Invocation (RMI)**. Through the use of the RmiProxyFactoryBean and the RmiServiceExporter Spring supports both traditional RMI (with java.rmi.Remote interfaces and java.rmi.RemoteException) and transparent remoting via RMI invokers (with any Java interface).
- **Spring's HTTP invoker**. Spring provides a special remoting strategy which allows for Java serialization via HTTP, supporting any Java interface (just like the RMI invoker). The corresponding support classes are HttpInvokerProxyFactoryBean and HttpInvokerServiceExporter.
- **Hessian**. By using Spring's HessianProxyFactoryBean and the HessianServiceExporter you can transparently expose your services using the lightweight binary HTTP-based protocol provided by Caucho.
- **Burlap**. Burlap is Caucho's XML-based alternative to Hessian. Spring provides support classes such as BurlapProxyFactoryBean and BurlapServiceExporter.
- **JAX-RPC**. Spring provides remoting support for web services via JAX-RPC (J2EE 1.4's web service API).
- **JAX-WS**. Spring provides remoting support for web services via JAX-WS (the successor of JAX-RPC, as introduced in Java EE 5 and Java 6).
- **JMS**. Remoting using JMS as the underlying protocol is supported via the JmsInvokerServiceExporter and JmsInvokerProxyFactoryBean classes.

The major advantage of RMI, compared to Hessian and Burlap, is serialization. Effectively, any serializable Java object can be transported without hassle. Hessian and Burlap have their own (de-)serialization mechanisms, but are HTTP-based and thus much easier to setup than RMI. Alternatively, consider Spring's HTTP invoker to combine Java serialization with HTTP-based transport.

org.springframework.remoting.rmi.RmiServiceExporter

RMI exporter that exposes the specified service as RMI object with the specified name. Such services can be accessed via plain RMI or via RmiProxyFactoryBean. Also supports exposing any non-RMI service via RMI invokers, to be accessed via RmiClientInterceptor / RmiProxyFactoryBean's automatic detection of such invokers.

With an RMI invoker, RMI communication works on the RmiInvocationHandler level, needing only one stub for any service. Service interfaces do not have to extend java.rmi.Remote or throw java.rmi.RemoteException on all methods, but in and out parameters have to be serializable.

RMI makes a best-effort attempt to obtain the fully qualified host name. If one cannot be determined, it will fall back and use the IP address. Depending on your network configuration, in some cases it will resolve the IP to the loopback address. To ensure that RMI will use the host name bound to the correct network interface, you should pass the java.rmi.server.hostname property to the JVM that will export the registry and/or the service using the "-D" JVM argument.

For example: -Djava.rmi.server.hostname=myserver.com

Using the RmiServiceExporter, we can expose the interface as RMI object. The interface can be accessed by using RmiProxyFactoryBean, or via plain RMI in case of a traditional RMI service. The RmiServiceExporter explicitly supports the exposing of any non-RMI services via RMI invokers.

JPG to PDF Converter For Mac - Unregis

org.springframework.remoting.rmi.RmiProxyFactoryBean

FactoryBean for RMI proxies, supporting both conventional RMI services and RMI invokers. Exposes the proxied service for use as a bean reference, using the specified service interface. Proxies will throw Spring's unchecked RemoteAccessException on remote invocation failure instead of RMI's RemoteException.

The service URL must be a valid RMI URL like "rmi://localhost:1099/myservice". RMI invokers work at the RmiInvocationHandler level, using the same invoker stub for any service. Service interfaces do not have to extend java.rmi.Remote or throw java.rmi.RemoteException. Of course, in and out parameters have to be serializable.

With conventional RMI services, this proxy factory is typically used with the RMI service interface. Alternatively, this factory can also proxy a remote RMI service with a matching non-RMI business interface, i.e. an interface that mirrors the RMI service methods but does not declare RemoteExceptions. In the latter case, RemoteExceptions thrown by the RMI stub will automatically get converted to Spring's unchecked RemoteAccessException.

NOTE When using RMI, it's not possible to access the objects through the HTTP protocol, unless you're tunneling the RMI traffic. RMI is a fairly heavy-weight protocol in that it supports full-object serialization which is important when using a complex data model that needs serialization over the wire. However, RMI-JRMP is tied to Java clients: It is a Java-to-Java remoting solution.

org.springframework.remoting.RemoteAccessException

Generic remote access exception. A service proxy for any remoting protocol should throw this exception or subclasses of it, in order to transparently expose a plain Java business interface.

When using conforming proxies, switching the actual remoting protocol e.g. from Hessian to Burlap does not affect client code. Clients work with a plain natural Java business interface that the service exposes. A client object simply receives an implementation for the interface that it needs via a bean reference, like it does for a local bean as well.

A client may catch RemoteAccessException if it wants to, but as remote access errors are typically unrecoverable, it will probably let such exceptions propagate to a higher level that handles them generically. In this case, the client code doesn't show any signs of being involved in remote access, as there aren't any remoting-specific dependencies.

Even when switching from a remote service proxy to a local implementation of the same interface, this amounts to just a matter of configuration. Obviously, the client code should be somewhat aware that it might be working against a remote service, for example in terms of repeated method calls that cause unnecessary roundtrips etc. However, it doesn't have to be aware whether it is actually working against a remote service or a local implementation, or with which remoting protocol it is working under the hood.

Following are its sub-classes →

- org.springframework.remoting.RemoteConnectFailureException
- org.springframework.remoting.RemoteInvocationFailureException
- org.springframework.remoting.RemoteLookupFailureException
- org.springframework.remoting.RemoteProxyFailureException

JPG to PDF Converter For Mac - Unregis

Integrating Struts and Spring

The advantages of integrating a Struts application into the Spring framework are manifold. First off, Spring was explicitly designed to resolve some of the real-world problems of JEE, such as complexity, poor performance, testability, and much more. Second, the Spring framework includes an AOP implementation that lets you apply aspect-oriented techniques to normal object-oriented code. Third, some might say that the Spring framework just handles Struts better than Struts handles itself. But that's a matter of opinion, so follow along as I demonstrate three approaches to integrating Struts applications into the Spring framework, and then decide for yourself.

The approaches I demonstrate are all relatively simple to execute but they offer distinctly different advantages. So that you can fully understand each approach, I've created a separate working example for each one. See the Download section for the complete example source code. See Resources to download Struts MVC and the Spring framework.

What's so great about Spring?

Spring creator Rod Johnson took a critical eye to Java™ Enterprise software development and suggested that many enterprise issues could be resolved by the strategic use of the IOC pattern, also known as dependency injection. When Rod and a dedicated team of open-source developers put his theories into practice, the result was the Spring framework. In a nutshell, Spring is a lightweight container that makes it easy to wire objects together using an external XML configuration file. Each object can receive a reference to a dependent object by exposing a JavaBean property, leaving you with the simple task of "wiring them up" in an XML configuration file.

IOC and Spring

IOC is a design pattern that externalizes application logic so that it can be injected into client code rather than written into it. Combining IOC with the use of programming to interfaces, as the Spring framework does, yields an architecture that reduces the client's dependency on implementation-specific logic. See Resources for more on IOC and Spring.

Dependency injection is a powerful feature, but the Spring framework offers much more. Spring supports pluggable transaction managers to give you a broader range of choices for transaction handling. It integrates leading persistence frameworks while also offering a consistent exception hierarchy. Spring also provides a simple mechanism for applying aspect-oriented code against normal, object-oriented code.

Spring AOP lets you use interceptors to intercept application logic at one or more execution points. Interceptors are widely used for for logging because consolidating an application's logging logic in interceptors results in a more readable, functional code base. As you'll soon see, Spring AOP ships with its own interceptors for addressing cross-cutting concerns and also lets you write your own.

Like Struts, Spring can also function as an MVC implementation. Both frameworks have their merits and drawbacks, although most would agree that Struts is still king when it comes to MVC. Many development teams have learned to rely on Struts as the foundation for building quality software under strict deadlines. With so much momentum behind Struts, even development teams that would like to integrate features of the Spring framework don't want to switch to Spring MVC. The good news is that you don't have to. The Spring architecture allows you to connect Struts as your Web framework to Spring-based business and persistence layers. The end result is that you can have your cake and eat it too!

In the recipes that follow, you'll learn three ways to integrate Struts MVC into the Spring framework. I'll expose the cons of each recipe as well as its comparative advantages. Once you've seen all three in action, I'll show you an exciting application of the approach I like best.

JPG to PDF Converter For Mac - Unregis

Three little recipes

Each of the following integration techniques (or recipes) has its merits, as well as its own particular quirks. I'm partial to only one of them, but knowing them all will deepen your understanding of both Struts and Spring. It will also provide you with a broad range of options for dealing with various scenarios. The recipes are as follows:

- Use Spring's ActionSupport class to integrate Struts
- Override the Struts RequestProcessor with Spring's DelegatingRequestProcessor
- Delegate Struts Action management to the Spring framework

Loading the application context

No matter which technique you use, you will need to use the Spring ContextLoaderPlugin to load the Spring application context for the Struts ActionServlet. Simply add the plug-in to your struts-config.xml file as you would any other plug-in, as shown here:

```
<plug-in className=
        "org.springframework.web.struts.ContextLoaderPlugIn">
    <set-property property=
        "contextConfigLocation" value="/WEB-INF/beans.xml"/>
</plug-in>
```

Recipe 1. Use Spring's ActionSupport

Creating a Spring context manually is the most intuitive way to integrate Struts with Spring. To make it even easier, Spring offers a little help. The org.springframework.web.struts.ActionSupport class provides a getWebApplicationContext() method to easily obtain a Spring context. All you need to do is extend your action from Spring's ActionSupport instead of the Struts Action class, as shown in Listing 1:

Listing 1. Using ActionSupport to integrate Struts

```
package ca.nexcel.books.actions;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.DynaActionForm;
import org.springframework.context.ApplicationContext;
import org.springframework.web.struts.ActionSupport;
import ca.nexcel.books.beans.Book;
import ca.nexcel.books.business.BookService;
```


JPG to PDF Converter For Mac - Unregis

```
public class SearchSubmit extends ActionSupport {   |(1)
  public ActionForward execute(
    ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response)
    throws IOException, ServletException {
        DynaActionForm searchForm = (DynaActionForm) form;
        String isbn = (String) searchForm.get("isbn");

        //the old fashion way
        //BookService bookService = new BookServiceImpl();

        ApplicationContext ctx =
          getWebApplicationContext();    |(2)
        BookService bookService =
          (BookService) ctx.getBean("bookService");   |(3)

        Book book = bookService.read(isbn.trim());
        if (null == book) {
          ActionErrors errors = new ActionErrors();
          errors.add(ActionErrors.GLOBAL_ERROR,new ActionError
            ("message.notfound"));
          saveErrors(request, errors);
          return mapping.findForward("failure") ;
        }
          request.setAttribute("book", book);
          return mapping.findForward("success");
      }
}
```

Let's quickly consider what's happening here. At (1), I create an Action by extending from the Spring ActionSupport class rather than the Struts Action class. At (2), I use the getWebApplicationContext() method to obtain an ApplicationContext. To obtain the business service, I use the context obtained at (2) to look up a Spring bean at (3).

This technique is simple and easy to understand. Unfortunately, it couples the Struts action to the Spring framework. If you ever decide to replace Spring, you would have to rewrite the code. Moreover, because the Struts action isn't under Spring's control, you can't reap the benefits of Spring AOP. This technique may be useful when using multiple independent Spring [illegible] for the most part it is not as desirable a solution as the other two choices.

JPG to PDF Converter For Mac - Unregis

Recipe 2. Override the RequestProcessor

Decoupling Spring from the Struts action is a much smarter design choice. One way to do this is to override the Struts RequestProcessor processor with the org.springframework.web.struts.DelegatingRequestProcessor class, as shown in Listing 2:

Listing 2. Integration via Spring's DelegatingRequestProcessor

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
    "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
<struts-config>
 <form-beans>
  <form-bean name="searchForm"
   type="org.apache.struts.validator.DynaValidatorForm">
        <form-property name="isbn"  type="java.lang.String"/>
  </form-bean>
  </form-beans>
 <global-forwards type="org.apache.struts.action.ActionForward">
   <forward  name="welcome"          path="/welcome.do"/>
   <forward  name="searchEntry"      path="/searchEntry.do"/>
   <forward  name="searchSubmit"     path="/searchSubmit.do"/>
 </global-forwards>
 <action-mappings>
   <action   path="/welcome" forward="/WEB-INF/pages/welcome.htm"/>
   <action   path="/searchEntry" forward="/WEB-INF/pages/search.jsp"/>
   <action   path="/searchSubmit"
        type="ca.nexcel.books.actions.SearchSubmit"
        input="/searchEntry.do"
        validate="true"
        name="searchForm">
        <forward name="success" path="/WEB-INF/pages/detail.jsp"/>
        <forward name="failure" path="/WEB-INF/pages/search.jsp"/>
   </action>
 </action-mappings>
 <message-resources parameter="ApplicationResources"/>
 <controller processorClass="org.springframework.web.struts.
 DelegatingRequestProcessor"/> |(1)
```


JPG to PDF Converter For Mac - Unregis

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property property="pathnames"
    value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
</plug-in>
<plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">
  <set-property property="csntextConfigLocation" value="/WEB-INF/beans.xml"/>
</plug-in>
</struts-config>
```

Here, I've used the <controller> tag to override the default Struts RequestProcessor with the DelegatingRequestProcessor. My next step is to register the action in my Spring config file, as shown in Listing 3:

Listing 3. Registering an action in the Spring config file

```
<?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="bookService" class="ca.nexcel.books.business.BookServiceImpl"/>
  <bean name="/searchSubmit"
    class="ca.nexcel.books.actions.SearchSubmit"> |(1)
      <property name="bookService">
        <ref bean="bookService"/>
      </property>
  </bean>
</beans>
```

Note that at (1), I've registered a bean using the name attribute to match the struts-config action mapping name. The SearchSubmit action exposes a JavaBean property, allowing Spring to populate the property at run time, as shown in Listing 4:

Listing 4. A Struts action with a JavaBean property

```
package ca.nexcel.books.actions;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
```


JPG to PDF Converter For Mac - Unregis

```
import org.apache.struts.action.DynaActionForm;
import ca.nexcel.books.beans.Book;
import ca.nexcel.books.business.BookService;
public class SearchSubmit extends Action {
 private BookService bookService;
 public BookService getBookService() {
  return bookService;
 }
 public void setBookService(BookService bookService) { | (1)
  this.bookService = bookService;
 }
 public ActionForward execute(
  ActionMapping mapping,
  ActionForm form,
  HttpServletRequest request,
  HttpServletResponse response)
  throws IOException, ServletException {
  DynaActionForm searchForm = (DynaActionForm) form;
  String isbn = (String) searchForm.get("isbn");
 Book book = getBookService().read(isbn.trim());  |(2)
  if (null == book) {
   ActionErrors errors = new ActionErrors();
   errors.add(ActionErrors.GLOBAL_ERROR,new ActionError("message.notfound"));
   saveErrors(request, errors);
   return mapping.findForward("failure") ;
  }
   request.setAttribute("book", book);
   return mapping.findForward("success");
  }
}
```

In Listing 4, you can see how to build the Struts action. At (1), I create a JavaBean property. This property is automatically populated by the DelegatingRequestProcessor. This design protects the Struts action from knowing it's being managed by Spring while giving you all the benefits of Spring's action management framework. Because your Struts actions are oblivious to the existence of Spring, you can swap out Spring for some other inversion of control container without refactoring your Struts code.

While the DelegatingRequestProcessor approach is definitely better than the first one, it does have some problems. If you were using a different RequestProcessor, then you would need to integrate the Spring DelegatingRequestProcessor manually. The added code would become a maintenance hassle and would also reduce your application's flexibility going forward. Moreover,

JPG to PDF Converter For Mac - Unregis

there has been some talk of replacing the Struts RequestProcessor with a chain of command. Such a change would negatively impact the longevity of this solution.

Recipe 3. Delegate action management to Spring

A much better solution is to delegate Struts action management to the Spring framework. You can do this by registering a proxy in the struts-config action mapping. The proxy is responsible for looking up the Struts action in the Spring context. Because the action is under Spring's control, it populates the action's JavaBean properties and leaves the door open to applying features such as Spring's AOP interceptors.

In Listing 5, the Action class is the same as it was in Listing 4. However, the struts-config is a little different:

Listing 5. The delegation method of Spring integration

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC
        "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
        "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
<struts-config>
 <form-beans>
   <form-bean name="searchForm"
    type="org.apache.struts.validator.DynaValidatorForm">
          <form-property name="isbn"   type="java.lang.String"/>
   </form-bean>
 </form-beans>
<global-forwards type="org.apache.struts.action.ActionForward">
    <forward   name="welcome"          path="/welcome.do"/>
    <forward   name="searchEntry"      path="/searchEntry.do"/>
    <forward   name="searchSubmit"     path="/searchSubmit.do"/>
 </global-forwards>
<action-mappings>
    <action    path="/welcome" forward="/WEB-INF/pages/welcome.htm"/>
    <action    path="/searchEntry" forward="/WEB-INF/pages/search.jsp"/>
    <action    path="/searchSubmit"
          type="org.springframework.web.struts.DelegatingActionProxy" |(1)
          input="/searchEntry.do"
          validate="true"
          name="searchForm">
          <forward name="success" path="/WEB-INF/pages/detail.jsp"/>
          <forward name="failure" path="/WEB-INF/pages/search.jsp"/>
    </action>
```


JPG to PDF Converter For Mac - Unregis

```
</action-mappings>
<message-resources parameter="ApplicationResources"/>
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property
  property="pathnames"
  value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
</plug-in>
<plug-in
  className="org.springframework.web.struts.ContextLoaderPlugIn">
  <set-property property="contextConfigLocation" value="/WEB-INF/beans.xml"/>
</plug-in>
</struts-config>
```

Listing 5 is a typical struts-config.xml file, except for one small difference. Instead of declaring the action's class name, it registers the name of Spring's proxy class, as shown at (1). The DelegatingActionProxy class uses the action mapping name to look up the action in the Spring context. This is the context that was declared with ContextLoaderPlugIn.

Registering a Struts action as a Spring bean is very straightforward, as shown in Listing 6. I simply create a bean using the name of the action mapping using the <bean> tag's name attribute (in this case, "/searchSubmit"). The action's JavaBean properties are populated like any Spring bean:

Listing 6. Register a Struts action in the Spring context

```
<?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="bookService" class="ca.nexcel.books.business.BookServiceImpl"/>
 <bean name="/searchSubmit"
    class="ca.nexcel.books.actions.SearchSubmit">
  <property name="bookService">
    <ref bean="bookService"/>
  </property>
 </bean>
</beans>
```

The benefits of action delegation

The action-delegation solution is the best of the three. The Struts action has no knowledge of Spring and could be used in non-Spring applications without changing a single line of code. It's not at the mercy of a change to the RequestProcessor, and it can take advantage of Spring's AOP features.

The benefits of action delegation don't stop there, either. Once you have your Struts action under Spring's control, you can leverage Spring to give them more pizzazz. For example, without Spring, all Struts actions must be threadsafe. If you set the <bean> tag's singleton attribute to

JPG to PDF Converter For Mac - Unregis

"false," however, your application will have a newly minted action object on every request. You might not need this feature, but it's nice to know you have it in your back pocket. You can also take advantage of Spring's lifecycle methods. For example, the <bean> tag's init-method attribute is used to run a method when the Struts action is instantiated. Similarly, the destroy-method attribute executes a method just before the bean is removed from the container. These methods are a great way to manage expensive objects in much the same way as the Servlet lifecycle does.

Intercepting Struts

As previously mentioned, one of the chief advantages of combining Struts and Spring, and doing it by delegating Struts actions to the Spring framework, is that you can apply Spring's AOP interceptors to your Struts actions. By applying Spring interceptors to Struts actions, you can tackle cross-cutting concerns with minimal effort.

Spring offers a few built-in interceptors, but I'll show you how to create your own and apply it to a Struts action. To use an interceptor, you need to do three things:

1. Create the interceptor.
2. Register it.
3. Declare where it will intersect the code.

This is pretty simple stuff but also very powerful. For example, in Listing 7, I create a logging interceptor for a Struts action. This interceptor prints out a statement before every method call:

Listing 7. A simple logging interceptor

```
package ca.nexcel.books.interceptors;
import org.springframework.aop.MethodBeforeAdvice;
import java.lang.reflect.Method;
public class LoggingInterceptor implements MethodBeforeAdvice {
  public void before(Method method, Object[] objects, Object o) throws Throwable {
      System.out.println("logging before!");
  }
}
```

This interceptor is very simple. The before() method is executed before every method in its intersection. In this case, it prints out a statement, but it could do anything you like. The next step is to register the interceptor in the Spring configuration file, shown in Listing 8:

Listing 8. Registering the interceptor in the Spring config file

```
<?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="bookService" class="ca.nexcel.books.business.BookServiceImpl"/>
  <bean name="/searchSubmit"
      class="ca.nexcel.books.actions.SearchSubmit">
    <property name="bookService">
      <ref bean="bookService"/>
    </property>
```


JPG to PDF Converter For Mac - Unregis

```
</bean>
<!-- Interceptors -->
<bean name="logger"
 class="ca.nexcel.books.interceptors.LoggingInterceptor"/> |(1)
<!-- AutoProxies -->
<bean name="loggingAutoProxy"
    class="org.springframework.aop.framework.autoproxy.
     BeanNameAutoProxyCreator"> |(2)
  <property name="beanNames">
     <value>/searchSubmit</valuesgt; |(3)
  </property>
  <property name="interceptorNames">
     <list>
      <value>logger</value> |(4)
     </list>
  </property>
  </bean>
</beans>
```

As you've probably noticed, Listing 8 extends the application shown in Listing 6 to include an interceptor. Details are as follows:

- At (1), I register the interceptor.
- At (2), I create a bean name autoproxy describing how the interceptor is applied. There are other ways to define intersections, but this approach is common and easy to do.
- At (3), I register the Struts action as the bean that will be intercepted. If you wanted to intersect other Struts actions, then you could simply create additional <value> tags under "beanNames."
- At (4), when the interception occurs, I execute the name of the interceptor bean created at (1). All the interceptors listed here are applied against the "beanNames."

That's it! As this example shows, putting your Struts actions under control of the Spring framework opens up a whole new set of options for handling your Struts applications. In the case of this example, action delegation makes it easy to utilize Spring interceptors for better logging in Struts applications.

Conclusion

In this article, you learned three recipes for integrating Struts actions into the Spring framework. Using Spring's ActionSupport to integrate Struts (as I did in the first recipe) is quick and easy but couples your Struts actions to the Spring framework. If you ever needed to port the application to a different framework you would need to rewrite the code. The second solution of delegating the RequestProcessor cleverly decouples your code, but it doesn't necessarily scale well and may not last long if the Struts RequestProcessor is revised to a chain of command. The third approach is the best of the three: delegating Struts actions to the Spring framework results in decoupled code that lets you utilize Spring's features (such as logging interceptors) in your Struts applications.

JPG to PDF Converter For Mac - Unregis

Aspect Oriented Programming

1) Introduction

- One of the major features available in the Spring Distribution is the provision for separating the ***cross-cutting concerns*** in an Application through the means of ***Aspect Oriented Programming***. Aspect Oriented Programming is sensibly new and it is not a replacement for ***Object Oriented Programming***. In fact, ***AOP*** is another way of organizing your Program Structure.

2.1) The Real Problem

Since AOP is relatively new, this section devotes time in explaining the need for ***Aspect Oriented Programming*** and the various terminologies that are used within. Let us look into the traditional model of before explaining the various concepts.

Consider the following sample application,

Account.java

```
public class Account{

  public long deposit(long depositAmount){

    newAmount = existingAmount + depositAmount;
    currentAmount = newAmount;
    return currentAmount;

  }

  public long withdraw(long withdrawalAmount){

    if (withdrawalAmount <= currentAmount){
      currentAmount = currentAmount - withdrawalAmount;
    }
    return currentAmount;

  }
}
```

The above code models a simple Account Object that provides services for deposit and withdrawal operation in the form of Account.deposit() and Account.withdraw() methods. Suppose say we want to add some bit of the security to the Account class, telling that only users with BankAdmin privilege is allowed to do the operations. With this new requirement being added let us see the modified class structure below.

JPG to PDF Converter For Mac - Unregis

Account.java

```
public class Account{

  public long deposit(long depositAmount){

    User user = getContext().getUser();
    if (user.getRole().equals("BankAdmin"){
      newAmount = existingAmount + depositAmount;
      currentAmount = newAmount;
    }
    return currentAmount;

  }

  public long withdraw(long withdrawalAmount){

    User user = getContext().getUser();
    if (user.getRole().equals("BankAdmin"){
      if (withdrawalAmount <= currentAmount){
        currentAmount = currentAmount - withdrawalAmount;
      }
    }
    return currentAmount;
  }
}
```

security service.

security service

Assume that getContext().getUser() someway gives the current User object who is invoking the operation. See the modified code mandates the use of adding additional if condition before performing the requested operation. Assume that another requirement for the above Account class is to provide some kind of ***Logging*** and ***Transaction Management*** Facility. Now the code expands as follows,

Account.java

```
public class Account{
  public long deposit(long depositAmount){
    logger.info("Start of deposit method");
    Transaction trasaction = getContext().getTransaction();
    transaction.begin();
```

logging service.

Transaction management service

JPG to PDF Converter For Mac - Unregis

```
    try{
        User user = getContext().getUser();
        if (user.getRole().equals("BankAdmin"){
            newAmount = existingAamount + depositAmount;
            currentAmount = newAmount;
        }
        transaction.commit();
    }catch(Exception exception){
        transaction.rollback();
    }
    logger.info("End of deposit method");
    return currentAmount;

}

public long withdraw(long withdrawalAmount){

    logger.info("Start of withdraw method");
    Transaction trasaction = getContext().getTransaction();
    transaction.begin();
    try{
        User user = getContext().getUser();
        if (user.getRole().equals("BankAdmin"){
            if (withdrawalAmount <= currentAmount){
                currentAmount = currentAmount - withdrawalAmount;
            }
        }
        transaction.commit();
    }catch(Exception exception){
        transaction.rollback();
    }
    logger.info("End of withdraw method");
    return currentAmount;

}
}
```

→ security service

→ Business logic

→ Transaction service

→ Transaction service

→ Logging service

JPG to PDF Converter For Mac - Unregis

The above code has so many dis-advantages. The very first thing is that as soon as new requirements are coming it is forcing the methods and the logic to change a lot which is against the ***Software Design***. Remember every piece of newly added code has to undergo the Software Development Lifecycle of Development, Testing, Bug Fixing, Development, Testing, ... This, certainly cannot be encouraged in particularly big projects where a single line of code may have multiple dependencies between other Components or other Modules in the Project.

2.2) The Solution through AOP

Let us re-visit the Class Structure and the Implementation to reveal the facts. The Account class provides services for depositing and withdrawing the amount. But when you look into the implementation of these services, you can find that apart from the normal business logic, it is doing so many other stuffs like ***Logging***, ***User Checking*** and ***Transaction Management***. See the pseudo-code below that explains this.

```
public void deposit(){

  // Transaction Management
  // Logging
  // Checking for the Privileged User
  // Actual Deposit Logic comes here

}

public void withdraw(){

  // Transaction Management
  // Logging
  // Checking for the Privileged User
  // Actual Withdraw Logic comes here

}
```

From the above pseudo-code, it is clear that ***Logging***, ***Transaction Management*** and ***User Checking*** which are never part of the Deposit or the Service functionality are made to embed in the implementation for completeness. Specifically, AOP calls this kind of logic that ***cross-cuts*** or overlaps the existing business logic as ***Concerns*** or ***Cross-Cutting Concerns***. The main idea of AOP is to isolate the cross-cutting concerns from the application code thereby modularizing them as a different entity. It doesn't mean that because the cross-cutting code has been externalized from the actual implementation, the implementation now doesn't get the required add-on functionalities. There are ways to specify some kind of relation between the original business code and the Concerns through some techniques which we will see in the subsequent sections.

3) AOP Terminologies

It is hard to get used with the ***AOP terminologies*** at first but a thorough reading of the following section along with the illustrated samples will make it easy. Let us look into the majorly used AOP jargons.

JPG to PDF Converter For Mac - Unregis

3.1) Aspects

An ***Aspect*** is a functionality or a feature that ***cross-cuts over*** objects. The addition of the functionality makes the code to ***Unit Test*** difficult because of its dependencies and the availability of the various components it is referring. For example, in the below example, ***Logging*** and ***Transaction Management*** are the aspects.

```
public void businessOperation(BusinessData data){

  // Logging
  logger.info("Business Method Called");

  // Transaction Management Begin
  transaction.begin();

  // Do the original business operation here

  transaction.end();
}
```

3.2) JoinPoint

Join Points defines the various ***Execution Points*** where an ***Aspect*** can be applied. For example, consider the following piece of code,

```
public void someBusinessOperation(BusinessData data){

  //Method Start -> Possible aspect code here like logging.

  try{
    // Original Business Logic here.
  }catch(Exception exception){
    // Exception -> Aspect code here when some exception is raised.
  }finally{
    // Finally -> Even possible to have aspect code at this point too.
  }

  // Method End -> Aspect code here in the end of a method.
}
```

In the above code, we can see that it is possible to determine the various points in the execution of the program like ***Start of the Method***, ***End of the Method***, the ***Exception Block***, the ***Finally Block*** where a particular piece of ***Aspect*** can be made to execute. Such ***Possible Execution Points*** in the Application code for embedding ***Aspects*** are called ***Join Points***. It is not necessary that an ***Aspect*** should be applied to all the possible ***Join Points***.

JPG to PDF Converter For Mac - Unregis

3.3) Pointcut

As mentioned earlier, ***Join Points*** refer to the ***Logical Points*** wherein a particular ***Aspect*** or a ***Set of Aspects*** can be applied. A ***Pointcut*** or a ***Pointcut Definition*** will exactly tell on which ***Join Points*** the ***Aspects*** will be applied. To make the understanding of this term clearer, consider the following piece of code,

```
aspect LoggingAspect {}

aspect TransactionManagementAspect {}
```

Assume that the above two declarations declare something of type Aspect. Now consider the following piece of code,

```
public void someMethod(){

  //Method Start

  try{
    // Some Business Logic Code.
  }catch(Exception exception){
    // Exception handler Code
  }finally{
    // Finally Handler Code for cleaning resources.
  }

  // Method End
}
```

In the above sample code, the possible execution points, i.e. ***Join Points***, are the start of the method, end of the method, exception block and the finally block. These are the possible points wherein any of the aspects, ***Logging Aspect*** or ***Transaction Management Aspect*** can be applied. Now consider the following ***Point Cut*** definition,

```
pointcut method_start_end_pointcut(){
  // This point cut applies the aspects, logging and transaction, before the
  // beginning and the end of the method.

}

pointcut catch_and_finally_pointcut(){
  // This point cut applies the aspects, logging and transaction, in the catch
  // block (whenever an exception raises) and the finally block.

}
```


JPG to PDF Converter For Mac - Unregis

As clearly defined, it is possible to define a ***Point Cut*** that binds the ***Aspect*** to a particular ***Join Point*** or some ***Set of Join Points***.

3.4) Advice

Now that we are clear with the terms like ***Aspects***, ***Point Cuts*** and ***Join Points***, let us look into what actually ***Advice*** is. To put simple, Advice is the code that implements the ***Aspect***. In general, an ***Aspect*** defines the functionality in a more abstract manner. But, it is this ***Advice*** that provides a ***Concrete code Implementation*** for the ***Aspect***.

4) Creating Advices in Spring

As mentioned previously, ***Advice*** refers to the actual implementation code for an ***Aspect***. Other Aspect Oriented Programming Languages also provide support for ***Field Aspect***, i.e. intercepting a field before its value gets affected. But Spring provides support only ***Method Aspect***. The following are the different types of aspects available in Spring.

- Before Advice
- After Advice
- Throws Advice
- Around Advice

4.1) Before Advice

Before Advice is used to intercept before the method execution starts. In AOP, Before Advice is represented in the form of org.springframework.aop.BeforeAdvice. For example, a System should make security check on users before allowing them to accessing resources. In such a case, we can have a Before Advice that contains code which implements the User Authentication Logic.

Consider the following piece of Code,

Authentication.java

```
public class Authentication implements BeforeAdvice{

    public void before(Method method, Object[] args, Object target) throws Throwable{

        if (args[0] instanceof User){
            User user = (User)args[0];
            // Authenticate if he/she is the right user.
        }

    }

}
```

The above class extends ***BeforeAdvice***, thereby telling that before() method will be called before the execution of the method call. Note that the java.lang.reflect.Method method object represents target method to be invoked, Object[] args refers to the various arguments that are passed on to the method and target refers to the object which is calling the method.

*1 → Reflection Api Method class object representing the business method on which these advice is applied.
*2 → represents the arguments values of business method.
*3 → represents the spring bean on which this advice is applied.

JPG to PDF Converter For Mac - Unregis

*1 → represents the return value given by business method.

System.java


```
public class System{

  public void login(){
    // Apply Authentication Advice here.
  }

  public void logout(){
    // Apply Authentication Advice here too.
  }
}
```

Note that, till now we have not seen any code that will bind the Advice implementation to the actual calling method. We will see how to do that in the Samples section.

4.2) After Advice

After Advice will be useful if some logic has to be executed before ***Returning the Control*** within a method execution. This advice is represented by the interface org.springframework.aop.AfterReturningAdvice. For example, it is common in Application to Delete the Session Data and the various information pertaining to a user, after he has logged out from the Application. These are ideal candidates for ***After Advice***.

CleanUpOperation.java

```
public class CleanUpOperation implements AfterReturningAdvice {

  public void afterReturning(Object returnValue, Method method, Object[] args,
  Object target) throws Throwable{
    // Clean up session and user information.
  }

}
```

*1 *2 *3 *4

Note that, afterReturning() will be method that will be called once the method returns normal execution. If some exception happens in the method execution the afterReturning() method will never be called.

4.3) Throws Advice

When some kind of exception happens during the execution of a method, then to handle the exception properly, ***Throws Advice*** can be used through the means of org.springframework.aop.ThrowsAdvice. Note that this interface is a ***marker interface*** meaning that it doesn't have any method within it. The method signature inside the ***Throws Advice*** can take any of the following form,

```
public void afterThrowing(Exception ex)

public void afterThrowing(Method method, Object[] args, Object target,
Exception exception)
```

→ gives the exception that is raised for the business method.

JPG to PDF Converter For Mac - Unregis

For example, in a File Copy program, if some kind of exception happens in the mid-way then the newly created target file has to be deleted as the partial content in the file doesn't carry any sensible meaning. It can be easily achieved through the means of Throws Advice.

DeleteFile.java

```
public class DeleteFile implements ThrowsAdvice{

    public void afterThrowing(Method method, Object[] args, Object target,
    IOException exception){

        String targetFileName = (String)args[2];
        // Code to delete the target file.
    }

}
```

Note that the above method will be called when an Exception, that too of type IOException is thrown by the File Copy Program.

4.4) Around Advice

This Advice is very different from the other types of Advice that we have seen before, because of the fact that, this Advice provides finer control whether the target method has to be called or not. Considering the above advices, the return type of the method signature is always void meaning that, the Advice itself cannot change the return arguments of the method call. But ***Around Advice*** can even change the return type, thereby returning a brand new object of other type if needed.

Consider the following code,

```
pubic void relate(Object o1, Object o2){

    o1.establishRelation(o2);

}
```

Assume that we have a method that provides an Association link between two objects. But before that, we have to we want to ensure that the type of the Objects being passed must conform to a standard, by implementing some interfaces. We can have this arguments check in the ***Around Advice*** rather than having it in the actual method implementation. The Around Advice is represented by org.aopalliance.intercept.MethodInterceptor.

ValidateArguments.java

```
public class ValidateArguments implements MethodInterceptor {

    public Object invoke(MethodInvocation invocation) throws Throwable {
```

Note; Transaction Management Service is always specified as Around Advice having ability to begin transaction at the beginning of the business method and having

JPG to PDF Converter For Mac - Unregis

logic to commit or rollback operations at the end of the business method.


```
    Object arguments [] = invocation.getArguments()
    if ((arguments[0] instanceof Parent) && (arguments[1] instanceof Child) ){
        Object returnValue = invocation.proceed();
        return returnValue;
    }
    throw new Exception ("Arguments are of wrong type");
  }
}
```

→ calls the business method.

This logic executes at the beginning of the business method.

→ executes at the end of the business method ✓

In the above code, the validation happens over the arguments to check whether they implement the right interface. It is important to make a call to MethodInvocation.proceed(), if we are happy with the arguments validation, else the target method will never gets invoked.

5) Creating Point Cuts in Spring

Point Cuts define where exactly the Advices have to be applied in various ***Join Points***. Generally they act as ***Filters*** for the application of various Advices into the real implementation. Spring defines two types of ***Point Cuts*** namely the ***Static*** and the ***Dynamic Point Cuts***. The following section covers only about the Static Point Cuts as Dynamic Point Cuts are rarely used.

5.1) The Point Cut Interface

Point Cuts in Spring are represented by org.springframework.aop.Pointcut. Let us look into the various components of this interface. Looking at the interface definition will have something like the following,

Pointcut.java

```
public interface Pointcut{

  ClassFilter getClassFilter()
  MethodMatcher getMethodMatcher()

}
```

The getClassFilter() method returns a ClassFilter object which determines whether the classObject argument passed to the matches() method should be considered for giving ***Advices***. Following is a typical implementation of the ClassFilter interface.

MyClassFilter.java

```
public class MyClassFilter implements ClassFilter{

  public boolean matches(Class classObject){

    String className = classObject.getName();
    // Check whether the class objects should be advised based on their name.
    if (shouldBeAdviced(className) ){
```


JPG to PDF Converter For Mac - Unregis

```
        return true;
    }
    return false;
}
}
```

The next interface in consideration is the MethodMatcher which will filter whether various methods within the class should be given ***Advices***. For example, consider the following code,

MyMethodMatcher.java

```
class MyMethodMatcher implements MethodMatcher{

  public boolean matches(Method m, Class targetClass){

    String methodName = m.getName();
    if (methodName.startsWith("get")){
        return true;
    }
    return false;
  }

  public boolean isRuntime(){
    return false;
  }

  public boolean matches(Method m, Class target, Object[] args);

    // This method wont be called in our case. So, just return false.
    return false;
  }
}
```

In the above code, we have 3 methods defined inside in MethodMatcher interface. The isRuntime() method should return true when we want to go for Dynamic Point cut Inclusion by depending on the values of the arguments, which usually happens at run-time. In our case, we can return false, which means that matches(Method method, Class target, Object[] args) wont be called. The implementation of the 2 argument matches() method essentially says that we want only the getter methods to be advised.

We have convenient concrete classes' implementation the ***Point Cut*** classes which are used to give advices to methods statically. They are

- NameMatchMethod Pointcut
- Regular Expression Pointcut

JPG to PDF Converter For Mac - Unregis

5.2) NameMatchMethod Pointcut

Here the name of the methods that are too be given advices can me directly mentioned in the Configuration File. A '*' represents that all the methods in the class should be given Advice. For example consider the following class,

MyClass.java - springBean

```
public class MyClass{
  public void method1(){}
  public void method2(){}
  public void getMethod1()
  public void getMethod2()
}
```

Suppose we wish that only the methods getMethod1() and getMethod2() should be given ***Advice*** by some aspect. In that case, we can have the following Configuration file that achieves this,

```
<bean id="getMethodsAdvisor"
  class="org.springframework.aop.support.NameMatchMethodPointcutAdvisor">
```

Predefined advisor class

```
  <property name="mappedName">
    <value>get*2</value>
  </property>
```

exact! → the advice will be applied only on one business method (getMethod11)

```
</bean>
```

The Expression get* tells that all method names starting with the method name get will be given ***Advices***. If we want all the methods in the MyClass to be adviced, then the 'value' tag should be given '*' meaning all the methods in the Class.

5.3) Regular Expression Pointcut

This kind is used if you want to match the name of the methods in the Class based on ***Regular Expression***. Spring distribution already comes with two supported flavors of Regular Expression namely ***Perl Regular Expression*** (represented by org.springframework.aop.support.Perl5RegexpMethodPointcut) and ***Jdk Regular Expression*** (represented by org.springframework.aop.support.JdkRegexpMethodPointcut).

Considering the following class,

MyClass.java springBean

```
public class MyClass{
  public void method1(){}
  public void method11(){}
  public void method2(){}
```


JPG to PDF Converter For Mac - Unregis

```
    public void getMethod1()

    public void getMethod11()

    public void getMethod2()
}
```

The Expression 'm*1' matches method1() and method11(), 'getMethod.' matches only getMethod1() and getMethod2(). The ***Configuration File*** should be populated with the following information for gaining this kind of support.

```
<bean id="regExpAdvisor"
    class="org.springframework.aop.support.RegExpPointcutAdvisor">

    <property name="pattern">
        <value>m*1<value>
    </property>

</bean>
```

method

regular expression to apply advices on method1, method11() business methods.

6) Sample Application

6.1) Introduction

Let is illustrate the various types of Advices (***Before Advice***, ***After Advice***, ***Throws Advice*** and ***Around Advice***) that we saw before in this sample Application. For this sample application let us define Adder Service which provides logic for adding two numbers. The various classes involved in Application along with the Advices in the subsequent sections.

6.2) Addder.java

This is the interface definition for the Add Service. The interface name is Adder and it has one single method called add() taking two arguments both of type int.

Adder.java

```
package net.javabeat.spring.aop.introduction.test;

public interface Adder {

    public int add(int a,int b);

}
```

6.3) AdderImpl.java

The implementation class for the Add Service. The logic is as simple as it returns the summation of the two numbers given as arguments. Note that, in the later section we will see how Advices get bound with this Implementation Class.

JPG to PDF Converter For Mac - Unregis

AdderImpl.java

```
package net.javabeat.spring.aop.introduction.test;

public class AdderImpl implements Adder {

  public int add(int a, int b){
    return a+b;
  }

}
```

6.4) Before Advice Implementation

This is the ***Before Advice*** for the ***Adder*** Implmentation class. This class implements the before() method in the MethodBeforeAdvice interface by simply outputting a message telling that this advice is called.

LogBeforeCallAdvice.java

```
package net.javabeat.spring.aop.introduction.test;

import java.lang.reflect.Method;
import org.springframework.aop.MethodBeforeAdvice;

public class LogBeforeCallAdvice implements MethodBeforeAdvice{

  public void before(Method method, Object[] args, Object target) {
    System.out.println("Before Calling the Method");
  }

}
```

6.5) After Advice Implementation

The After Method Call Advice implements the AfterReturningAdvice interface providing implementation for the afterReturning() method. Like the ***Before Advice*** implementation, this Advice also outputs a simple message to the console.

LogAfterReturningAdvice.java

```
package net.javabeat.spring.aop.introduction.test;

import java.lang.reflect.Method;
import org.springframework.aop.AfterReturningAdvice;
```


JPG to PDF Converter For Mac - Unregis

```
public class LogAfterReturningAdvice implements AfterReturningAdvice{

  public void afterReturning(Object returnValue, Method method, Object[] args,
  Object target) throws Throwable {

    System.out.println("After Normal Return from Method");
  }

}
```

6.6) Throws Advice Implementation

This Advice will be called when some kind of Exception is caught during the method invocation. We have added a simple logic to simlate the exception when the user inputs are 0 and 0.

LogAfterThrowsAdvice.java

```
package net.javabeat.spring.aop.introduction.test;

import java.lang.reflect.Method;
import org.springframework.aop.ThrowsAdvice;

public class LogAfterThrowsAdvice implements ThrowsAdvice{

  public void afterThrowing(Method method, Object[] args, Object target,
  Exception exception){

    System.out.println("Exception is thrown on method " + method.getName());
  }

}
```

6.7) Around Advice Implementation

This Advice takes the entire control during the ***Method Execution***. It decides whether the add() method should be called or not based on the user inputs. Note that, only if the user inputs are not 0 and 0, then the add() method will be called through MethodInvocation.proceed().

LogAroundAdvice.java

```
package net.javabeat.spring.aop.introduction.test;

import org.aopalliance.intercept.*;
```


JPG to PDF Converter For Mac - Unregis

```
public class LogAroundAdvice implements MethodInterceptor{

  public Object invoke(MethodInvocation methodInvocation) throws Throwable {

    Object arguments[] = methodInvocation.getArguments();
    int number1 = ((Integer)arguments[0]).intValue();
    int number2 = ((Integer)arguments[1]).intValue();

    if (number1 == 0 && number2 == 0){
      throw new Exception("Dont know how to add 0 and 0!!!");
    }
    return methodInvocation.proceed();
  }
}
```

6.8) Configuration File

The Configuration File has 3 sections. One section is the ***Advice Bean Definition Section*** which is the definition set for all the 4 advices which we saw before. All the advices are given identifiers like 'beforeCall', 'afterCall', 'throwCall' and 'aroundCall'. Then contains the Bean Definition for the Add implementation class which is giving the identifier 'adderImpl'.

The next interesting section is how to ***bind these advices to the implementation code***. For this, we have to depend on ***ProxyFactory Bean***. This Bean is used to create Proxy objects for the Add Implementation class along with the Advice implementation. Note that the property 'proxyInterfaces' contains the Interface Name for which the proxy class has to ge generated. In our case, it is going to be the Adder interface. The 'interceptorNames' property takes a list of Advices to be applied to the dynamically generated proxy class. We have given all the 4 advices to this property. Finally the implementation class for the Adder service is given in the 'target' property.

aop-test.xml

```
<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">

  <!-- Advices -->
  <bean id = "beforeCall"
  class = "net.javabeat.spring.aop.introduction.test.LogBeforeCallAdvice" />

  <bean id = "afterCall"
  class = "net.javabeat.spring.aop.introduction.test.LogAfterReturningAdvice" />

  <bean id = "throwCall"
```


JPG to PDF Converter For Mac - Unregis

```
class = "net.javabeat.spring.aop.introduction.test.LogAfterThrowsAdvice" />

<bean id = "aroundCall"
class = "net.javabeat.spring.aop.introduction.test.LogAroundAdvice" />

<!-- Implementation Class -->
<bean id = "adderImpl"
class = "net.javabeat.spring.aop.introduction.test.AdderImpl" />

<!-- Proxy Implementation Class -->
<bean id = "adder"
class = "org.springframework.aop.framework.ProxyFactoryBean">

    <property name = "proxyInterfaces">
        <value>net.javabeat.spring.aop.introduction.test.Adder</value>
    </property>

    <property name = "interceptorNames">
        <list>
            <value>beforeCall</value>
            <value>afterCall</value>
            <value>throwCall</value>
            <value>aroundCall</value>
        </list>
    </property>

    <property name = "target">
        <ref bean = "adderImpl"/>
    </property>
  </bean>

</beans>
```

6.9) Test Class

Following is the test class for the Adder Service. The code loads the ***Bean Definition File*** by depending on the BeanFactory class. Watch carefully in the output for the various ***Advices*** getting called. Also, we have made to activate the ***Simulated Exception*** by passing 0 and 0 as arguments to the add() method call thereby making use of the ***Throws Advice***.

JPG to PDF Converter For Mac - Unregis

AdderTest.java

```
package net.javabeat.spring.aop.introduction.test;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.*;

public class AdderTest {

  public static void main(String args[]){

    Resource resource = new FileSystemResource("./src/aop-test.xml");
    BeanFactory factory = new XmlBeanFactory(resource);
    Adder adder = (Adder)factory.getBean("adder");
    int result = adder.add(10,10);
    System.out.println("Result = " + result);

    result = adder.add(0,0);
    System.out.println("Result = " + result);
  }
}
```

7) Conclusion

This article provided information on how to use ***Spring AOP*** for programming the ***Aspects*** in an Application. It started with defining what ***Aspects*** are and what are problems in having the ***Aspects*** directly embedded into the Application and how to separate them using ***AOP***. It then looked briefly into the various ***AOP Terminologies*** like ***Advice***, ***Point Cut***, ***Join Points*** etc. Then it moved on into the various support for creating ***Advices*** using ***Spring AOP***. Also covered in brief are the ***Static Point Cuts*** like ***Name Method Match*** and ***Regular Expression Point Cuts***. Finally the article concluded with a Sample Application that illustrates the usage of different Advices.

JPG to PDF Converter For Mac - Unregis

```
==========================================================================
App 58)>>>>>>>>>>>Demonstrates the behaviour of the AroundAdvice<<<<<<<<<<<<<<<<<<<
==========================================================================
---------------------------IBusinessLogic.java-----------------------
public interface IBusinessLogic
{
  public void method1();
}
-------------------------------BusinessLogic.java-----------------------------
public class BusinessLogic implements IBusinessLogic
{
    public void method1()
    {
      System.out.println("");
      System.out.println("execution of method1");
      System.out.println("");
    }
}
---------------------------------------AroundAdvice.java-----------------------------------
import org.aopalliance.intercept.MethodInvocation;
import org.aopalliance.intercept.MethodInterceptor;
public class AroundAdvice implements MethodInterceptor
{
  public Object invoke(MethodInvocation i1) throws Throwable
  {
    System.out.println("good evening everybody..")
    i1.proceed();
    System.out.println("Goodbye!");
    return null;
    }
  }
-----------------------------springconfig.xml-----------------------------
<beans>
  <bean id="pfb" class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="proxyInterfaces">
      <value>IBusinessLogic</value>
    </property>
    <property name="target">
      <ref local="targetobj"/>
    </property>
    <property name="interceptorNames">
      <list>
        <value>advr</value>
      </list>
    </property>
  </bean>
  <bean id="targetobj" class="BusinessLogic"/>

  <bean id="advr" class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
    <property name="advice">
      <ref local="adv"/>
    </property>
    <property name="pattern">
      <value>.*</value>
    </property>
  </bean>

  <bean id="adv" class="AroundAdvice"/>
</beans>
```

declaration of business method.

represents the executing one of business method

Advice class

mandatory to make your class as Around advice class

executes before executing (invoking) b. method

calls the business method

executes after invoking b. method.

spring configuration file

spring interface

ref. 47

ref. 49

gives proxy object by enabling advice on the target object.

our spring bean config

ref. 58

indicates this advice should be applied on all the business methods

our advice class configuration

JPG to PDF Converter For Mac - Unregis

client App

```
----------------------------MainApplication.java-----------------------------
import org.springframework.context.ApplicationContet;
import org.springframework.context.support.FileSystemXmlApplicationContext;
public class MainApplication
{
  public static void main(String[]args)
  {
    ApplicationContext ctx = new FileSystemXmlApplicationContext("springconfig.xml");
    IBusinessLogic proxyobj=(IBusinessLogic) ctx.getBean("businesslogicbean");
    proxyobj.method1()
    }
}
```

pfb
↳ ref 34

```
-----------------------------------------------------------------------------
==========To change business method return value using Around Advice==========
App59 (Aop)>>>>>>>>>>>>>>>>>>>>>>>>Based on Return Value<<<<<<<<<<<<<<<<<<<<<<<<
=============================================================================
-------------------------------IBusinessLogic.java-----------------------
public interface IBusinessLogic
{
  public int method1();
}
```

↳ spring-interface
→ business method with return type

```
-----------------------------------BusinessLogic.java-------------------
public class BusinessLogic implements IBusinessLogic
{
    public int method1()
    {
      System.out.println("");
      System.out.println("execution of method1");
      System.out.println("");
      return 1000;
    }
}
```

→ business method returning a value

```
----------------------------AroundAdvice.java-----------------------------
import org.aopalliance.intercept.MethodInvocation;
import org.aopalliance.intercept.MethodInterceptor;
public class AroundAdvice implements MethodInterceptor
{
  public Object invoke(MethodInvocation i1) throws Throwable
  {
    System.out.println("good evening everybody..");
    int
    Integer n=((Integer)i1.proceed()).intValue();
    System.out.println("Goodbye! ");
    if(n  <=100)
      return new Integer(0);
    else
      return new Integer(1);
  }
}
```

→ mandatory to develop advice as AroundAdvice

```
-----------------------------------------springconfig.xml-----------------------------------------
<?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="pfb" class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="proxyInterfaces">
      <value>IBusinessLogic</value>
    </property>
    <property name="target">
      <ref local="beanTarget"/>
    </property>
```


JPG to PDF Converter For Mac - Unregis

```
    <property name="interceptorNames">
      <list>
        <value>theAroundAdvisor</value>
      </list>
    </property>
  </bean>
  <bean id="beanTarget" class="BusinessLogic"/>

  <bean id="theAroundAdvisor"
  class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
    <property name="advice">
      <ref local="theAroundAdvice"/>
    </property>
    <property name="pattern">
      <value>.*</value>
    </property>
  </bean>

  <bean id="theAroundAdvice" class="AroundAdvice"/>
</beans>
-----------------------------MainAppliation.java--------------------------
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
public class MainApplication
{
  public static void main(String [] args)
  {
    ApplicationContext ctx = new FileSystemXmlApplicationContext("springconfig.xml");
    IBusinessLogic proxyobj = (IBusinessLogic) ctx.getBean("businesslogicbean");
    int n=proxyobj.method1();
    System.out.println("test result :"+n);
  }
}
-----------------------------------------------------------------------------
==============================================================================
App 60) >>>>>>>>>>>>>>>>>>>>>Based on argument values<<<<<<<<<<<<<<<<<<<<<<<
==============================================================================
---------------------------IBusinessLogic.java-------------------------
public interface IBusinessLogic
{
  public void method1(int id,String name);
}
-----------------------------------BusinessLogic.java-----------------------------
public class BusinessLogic implements IBusinessLogic
{
    public void method1(int id,String name)
    {
      System.out.println("");
      System.out.println("execution of method1");
      System.out.println(" id :"+id);
      System.out.println("name :"+name);
      System.out.println("");
    }
}
-----------------------------AroundAdvice.java-----------------------------
import org.aopalliance.intercept.MethodInvocation;
import org.aopalliance.intercept.MethodInterceptor;

public class AroundAdvice implements MethodInterceptor
{
  public Object invoke(MethodInvocation i1) throws Throwable
```

To change business method arguement values.

JPG to PDF Converter For Mac - Unregis

```
  {
    System.out.println("good evening everybody..");

 int x=((Integer)i1.getArguments()[0]).intValue();
  String y=(String)i1.getArguments()[1];
  if( x <= 0 )
     i1.getArguments()[0]=new Integer(101);
  if( y.length() <= 3)
     i1.getArguments()[1]="durga soft";

   i1.proceed();

    System.out.println("Goodbye! ");

    return null;
  }
}

----------springconfig.xml----------
<?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="pfb" class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="proxyInterfaces">
      <value>IBusinessLogic</value>
    </property>
    <property name="target">
      <ref local="beanTarget"/>
    </property>
    <property name="interceptorNames">
      <list>
        <value>advisor</value>
      </list>
    </property>
  </bean>
  <bean id="beanTarget" class="BusinessLogic"/>

  <bean id="advisor" class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
    <property name="advice">
      <ref local="ad"/>
    </property>
    <property name="pattern">
      <value>.*</value>
    </property>
  </bean>

  <bean id="ad" class="AroundAdvice"/>
</beans>
==========MainApplication.java==========
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
public class MainApplication
{
  public static void main(String [] args)
  {
    ApplicationContext ctx = new FileSystemXmlApplicationContext("springconfig.xml");
    IBusinessLogic proxyobj = (IBusinessLogic) ctx.getBean("proxyobj");
    proxyobj.method1(22,"ramesh");
    proxyobj.method1(-2,"rao");
  }
```

① gives first arguement value of business method

② gives second arguement value of business method.

changing first arguement value.

changing second argue. value

pfb → rao. 205

JPG to PDF Converter For Mac - Unregis

```
}
-----------------------------------------------------------------------------
=============================================================================
App 61)>>Application that configures multiple advisors on the business methods of spring bean
<<
=============================================================================
-----------------------------IBusinessLogic.java-----------------------------
public interface IBusinessLogic
{
  public void method1();
}
-----------------------------------BusinessLogic.java-----------------------------
public class BusinessLogic implements IBusinessLogic
{
   public void method1()
   {
     System.out.println("");
     System.out.println("execution of method1");
     System.out.println("");

   }
}
----------------------------BeforeAdvice1.java----------------------------
import org.springframework.aop.*;
import java.lang.reflect.*;
public class BeforeAdvice1 implements MethodBeforeAdvice
{

 public void before(Method method, Object[] args, Object target) throws Throwable
 {
  System.out.println("before advice");
 }

}
--------------------------AroundAdvice2.java-----------------------------
import org.aopalliance.intercept.MethodInvocation;
import org.aopalliance.intercept.MethodInterceptor;
public class AroundAdvice2 implements MethodInterceptor
{
  public Object invoke(MethodInvocation i1) throws Throwable
  {
    System.out.println(" AroundAdvice : good evening everybody..");

    i1.proceed();

    System.out.println("AroundAdvice : Goodbye! ");

  return null;

  }
}

----------------------------springconfig.xml----------------------------
<?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="pfb" class="org.springframework.aop.framework.ProxyFactoryBean">
    <property name="proxyInterfaces">
      <value>IBusinessLogic</value>
    </property>
```


JPG to PDF Converter For Mac - Unregis

```
    <property name="target">
      <ref local="targetobj"/>
    </property>
    <property name="interceptorNames">
      <list>
        <value>advr1</value>
        <value>advr2</value>
      </list>
    </property>
  </bean>

  <bean id="targetobj" class="BusinessLogic"/>

  <bean id="advr1"
  class="org.springframework.aop.support.NameMatchMethodPointcutAdvisor">
    <property name="advice">
      <ref local="adv1"/>
    </property>
    <property name="mappedName">
      <value>method1</value>
    </property>
  </bean>

<bean id="advr2" class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
    <property name="advice">
      <ref local="adv2"/>
    </property>
    <property name="pattern">
      <value>.*</value>
    </property>
  </bean>

  <bean id="adv1" class="BeforeAdvice1"/>
  <bean id="adv2" class="AroundAdvice2"/>
</beans>
----------------------------MainApplication.java----------------------------
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
public class MainApplication
{
  public static void main(String [] args)
  {
    ApplicationContext ctx = new FileSystemXmlApplicationContext("springconfig.xml");
    IBusinessLogic testObject = (IBusinessLogic) ctx.getBean("pfb");
    testObject.method1();
  }
}
-----------------------------------------------------------------------------
==============================================================================
App 61)>>>>>>Example application that deals with all the 4 types of advices<<<<<<<<<<<<<<
==============================================================================
----------------------------Adder.java----------------------------
package aop;
public interface Adder {
  public int add(int a,int b);
}
----------------------------Adderimpl.java----------------------------
package aop;

public class Adderimpl implements Adder {
```


JPG to PDF Converter For Mac - Unregis

```
    public int add(int a, int b){
        return a+b;
    }

}
```

spring configuration file.

----------aop-test.xml----------

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
    <!-- Advices -->
    <bean id = "beforeCall" class = "aop.LogBeforeCallAdvice" />
    <bean id = "afterCall" class = "aop.LogAfterReturningAdvice" />
    <bean id = "throwCall" class = "aop.LogAfterThrowsAdvice" />
    <bean id = "aroundCall" class = "aop.LogAroundAdvice" />

    <!-- Implementation Class(SpringBean) -->
    <bean id = "adderImpl" class = "aop.AdderImpl" />

    <!-- Proxy Implementation Class -->
    <bean id = "pfb" class = "org.springframework.aop.framework.ProxyFactoryBean">
        <property name = "proxyInterfaces">
            <value>aop.Adder</value>
        </property>
        <property name = "interceptorNames">
            <list>
                <value>beforeCall</value>
                <value>afterCall</value>
                <value>throwCall</value>
                <value>aroundCall</value>
            </list>
        </property>
        <property name = "target">
            <ref bean = "adderImpl"/>
        </property>
    </bean>
</beans>
```

Advices configuration

spring bean configuration

spring interface.

Here the advises are specified directly without advisor support that means all the advisors will be applied on the business methods.

ref 379

After advice.

----------LogAftgerReturningAdvice.java----------

```
package aop;

import java.lang.reflect.Method;
import org.springframework.aop.AfterReturningAdvice;
import org.apache.log4j.*;

public class LogAfterReturningAdvice implements AfterReturningAdvice{
  Logger logger = Logger.getLogger(LogAfterReturningAdvice.class);

  public   LogAfterReturningAdvice()
  {
  SimpleLayout layout = new SimpleLayout();
  FileAppender appender = null;
  try
  {
    appender = new FileAppender(layout,"output1.txt",false);
  }
  catch(Exception e) {}
   logger.addAppender(appender);
   logger.setLevel((Level) Level.DEBUG);
  }
```

log4j api

constructor

logfile

true allows append operations

specifies logger level to retrieve the log messages

JPG to PDF Converter For Mac - Unregis

```
@override
public void afterReturning(Object returnValue, Method method, Object[] args,
 Object target) throws Throwable {
    logger.info("After Normal Return from Method");
 }
}
```

log message

→ logging operation on business method through AfterAdvice class

----------LogAfterThrowsAdvice.java----------

→ ThrowsAdvice

```
package aop;

import java.lang.reflect.Method;
import org.springframework.aop.ThrowsAdvice;

public class LogAfterThrowsAdvice implements ThrowsAdvice{

  public void afterThrowing(Method method, Object[] args, Object target,
  Exception exception){

    System.out.println("Exception is thrown on method " + method.getName());
  }

}
```

→ It is recommended to use log4j here

----------LogAroundAdvice.java----------

→ Around Advice.

```
package aop;

import org.aopalliance.intercept.*;
public class LogAroundAdvice implements MethodInterceptor{

  public Object invoke(MethodInvocation i1) throws Throwable {

    Object arguments[] = i1.getArguments();
    int number1 = ((Integer)arguments[0]).intValue();
    int number2 = ((Integer)arguments[1]).intValue();

    if (number1 == 0 && number2 == 0){
       throw new Exception("Dont know how to add 0 and 0!!!");
    }
    return i1.proceed();
  }
}
```

reads the actual values of business method

calls business method

The ThrowsAdvice configured on the business method executes when exception is thrown in business method or when exception is thrown in the advices that are configured on the business method.

----------LogBeforeCallAdvice.java----------

→ Before Advice.

```
package aop;

import java.lang.reflect.Method;
import org.springframework.aop.MethodBeforeAdvice;
import org.apache.log4j.*;

public class LogBeforeCallAdvice implements MethodBeforeAdvice{

 Logger logger = Logger.getLogger(LogBeforeCallAdvice.class);

 public   LogBeforeCallAdvice()
 {
 SimpleLayout layout = new SimpleLayout();
 FileAppender appender = null;
  try
  {
    appender = new FileAppender(layout,"output1.txt",false);
  }
  catch(Exception e) {}
  logger.addAppender(appender);
 logger.setLevel((Level) Level.DEBUG);
```

→ throws exception for invalid method arguments. If this exception, the ThrowsAdvice will be executed.

→ logger file true.

JPG to PDF Converter For Mac - Unregis

```
483   }
484
485   public void before(Method method, Object[] args, Object target) {
486     logger.info("Before Calling the Method");
487   }
488 }
489
490 ----------------------------AopTestClient.java----------------------------
491 package aop;
492
493 import org.springframework.beans.factory.BeanFactory;
494 import org.springframework.beans.factory.xml.XmlBeanFactory;
495 import org.springframework.core.io.*;
496
497 public class AopTestClient {
498   public static void main(String args[]){
499
500       Resource resource = new FileSystemResource("aop-test.xml");
501       BeanFactory factory = new XmlBeanFactory(resource);
502       Adder adder = (Adder)factory.getBean("pfb");
503       int result = adder.add(10,5);
504       System.out.println("Result = " + result);
505       result = adder.add(10,0);
506       System.out.println("Result = " + result);
507   }
508 }
509
510 ==========================================================================
511 App 62)(AspectJ With schema support for before and after advices)
512 ==========================================================================
513 --------FooService.java----------
514 public interface FooService {
515 FooService getFoo(String fooName,int age);
516 void getAfter();
517 void getBefore(String myName);
518 }
519 --------DefaultFooService.java----------
520 public class DefaultFooService implements FooService {
521       public FooService getFoo(String fooName, int age) {
522             System.out.println("getFoo(fooName,age) business logic");
523             return null;
524       }
525       public void getAfter() {
526             System.out.println("getAfter() business logic");
527       }
528       final public void getBefore(String myName) {
529             System.out.println("getBefore(myName) business logic");
530       }
531 }
532 --------SimpleProfiler.java----------
533 public class SimpleProfiler {
534 public void afterMethod() throws Throwable {
535 System.out.println("After the method call");
536 }
537 public void beforeMethod(String myName){
538 System.out.println("My name is "+myName);
539 }
540 }
541 ---------spring.xml----------
542 <beans xmlns="http://www.springframework.org/schema/beans"
543 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```


JPG to PDF Converter For Mac - Unregis

#2 → The logic of before method belongs to SimpleProfiler class will be applied on getBefore-symbol method of Bean class

```
xmlns:aop="http://www.springframework.org/schema/aop"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">
<!-- this is the object that will be proxied by Spring's AOP infrastructure -->
<bean id="fooService" class="DefaultFooService"/>

<!-- this is the actual advice itself -->
<bean id="profiler" class="SimpleProfiler"/>

<aop:config>
<aop:aspect ref="profiler">
 <aop:pointcut id="aopafterMethod"  expression="execution(* FooService.*(..))"/>
  <aop:after pointcut-ref="aopafterMethod"  method="afterMethod"/>
  <aop:pointcut id="aopBefore" expression="execution(* FooService.getBefore(String)) and
  args(myName)"/>
  <aop:before pointcut-ref="aopBefore" method="beforeMethod"/>
 </aop:aspect>
</aop:config>
</beans>
```

→ Spring bean configuration
→ Advices configuration
→ ref 553
→ AOP
→ OGNL expression to apply pointing to all the methods of FooService interface
→ ref 534
→ ref 559
→ Here the logic of AfterMethod() belongs to SimpleProfiler class will be applied on all the business methods of Spring Bean class
→ pointcut pointing to getBefore-method
→ business methods
ref 537

```
----------client.java----------
import org.springframework.beans.factory.*;
import org.springframework.context.*;
import org.springframework.context.support.*;
public class client {
  public static void main(final String[] args) throws Exception {
    BeanFactory ctx = new ClassPathXmlApplicationContext("spring.xml");
    FooService foo = (FooService) ctx.getBean("fooService");
    foo.getFoo("raja", 12);
    foo.getAfter();
    foo.getBefore("raja");
  }
}
```

==========

App 63)(AspectJ With schema support for throws and around advices)

==========

```
----------TestInter.java----------
public interface TestInter
{
  void  demo(String name);
  long findfact(int k);
}
----------TestBean.java----------
public class TestBean implements TestInter
{
 public void demo(String name)
  {
      int k =Integer.parseInt(name);
  }
 public long findfact(int k)
 {
      long f=1;
      for(int i=1; i<=k; i++)
      {
      f = f * i;
      }
      return f;

  }
}
```

→ Spring interface
→ declaration of business methods

1. Spring jar
2. Commons-logging.jar
3. AspectJ rt.jar
AspectJ weaver.jar

JPG to PDF Converter For Mac - Unregis

class having throwsadvice, aroundadvice logic

```
----------MyAdvices.java----------
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.JoinPoint;
public class MyAdvices {
public void throwsMethod(JoinPoint jp,NumberFormatException nfe) throws Throwable
{
    System.out.println("this advice is applied for "+jp.getSignature().getName());
    System.out.println("The exception occured is : "+nfe.getMessage());
    System.out.println("advice from throws method");
}

public Object aroundMethod(ProceedingJoinPoint pjp) throws Throwable
{
    System.out.println("this advice is applied for "+pjp.getSignature().getName());
    long x = System.currentTimeMillis();
    Object retval=pjp.proceed();
    long y = System.currentTimeMillis();
    System.out.println("The method execution taken "+(y-x)+" milliseconds");
    System.out.println("The above service is from around advice");
    return retval;
 }
}
```

represents the business related info.

represents the exception that is raised in the business method

throwsadvice

acts as Around Advice

calls the business method.

all the are user defined methods, but signature is fixed (as per SEC)

```
----------spring.xml----------
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

 <bean id="id1" class="TestBean"/>
 <bean id="id2" class="MyAdvices"/>
 <aop:config>
  <aop:aspect ref="id2">
   <aop:pointcut id="pt1" expression="execution(* TestInter.demo(..))"/>
   <aop:after-throwing pointcut-ref="pt1" method="throwsMethod" throwing="nfe"/>

   <aop:pointcut id="pt2" expression="execution(long TestInter.*(..))"/>
   <aop:around pointcut-ref="pt2" method="aroundMethod"/>

  </aop:aspect>
 </aop:config>
</beans>
```

spring bean configuration

advices conf

pointcut pointing to demo method

ref 636

ref 639

608

608

650

point cut pointing to all the business methods

*1 → the logic of throwsAdvice belongs to MyAdvices class will be applied as throws advice on demo() business method.

type is long (so apt to findfact only)

```
----------client.java----------
import org.springframework.beans.factory.*;
import org.springframework.context.*;
import org.springframework.context.support.*;
public class client {
  public static void main(final String[] args) {
    BeanFactory ctx = new ClassPathXmlApplicationContext("spring.xml");
    TestInter ti=(TestInter)ctx.getBean("id1");
    try
    {
     ti.demo("durga");
    }
    catch(Exception e)
    {}
    System.out.println("---------");

    long m1 = ti.findfact(20);
```

This raises exception so throws advice will be executed

*2 → here the logics belongs to MyAdvice class) will be executed as aroundadvice on all the business methods

JPG to PDF Converter For Mac - Unregis

```
    System.out.println(m1);
    System.out.println("----------");

    long m2 = ti.findfact(58);
    System.out.println(m2);

  }
}
==========================================================
App 64)>>>>>>>>>>>>>>>>> AspectJ  Annotations<<<<<<<<<<<<<<<<<<<<<<<<
==========================================================
------------TestInter.java--------------------
public interface TestInter
{
      void  demo(String name);
      long findfact(int k);
}
------------TestBean.java--------------------
public class TestBean implements TestInter
{
public void demo(String name)
{
      int k =Integer.parseInt(name);
}
public long findfact(int k)
{
      long f=1;
      for(int i=1; i<=k; i++)
      {
            f = f * i;
      }
  return f;
  }
}
------------MyAspect.java----------------------
import org.aspectj.lang.annotation.*;
@Aspect
public class MyAspect
{
@Pointcut("execution(* TestInter.*(..))")
private void testinterOk(){}

@Before("testinterOk()")
public void beforeMethod()
{
 System.out.println("i am before advice");
}

 @AfterReturning("testinterOk()")
 public void afterReturningMethod()
 {
  System.out.println("i am after returning advice");
 }
}
------------spring.xml--------------------------
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:aop="http://www.springframework.org/schema/beans
xsi:schemaLocation="http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop
```

It makes this method as afteradvice

JPG to PDF Converter For Mac - Unregis

```
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

<bean id="id1" class="TestBean"/>
 <bean class="MyAspect"/>
 <aop:aspectj-autoproxy/>
 </beans>
-----------client.java-------------------------
import org.springframework.beans.factory.*;
import org.springframework.context.*;
import org.springframework.context.support.*;
public class client {
   public static void main(final String[] args) {
      BeanFactory ctx = new ClassPathXmlApplicationContext("spring.xml");
      TestInter ti=(TestInter)ctx.getBean("id1");
      try
      {
      ti.demo("durga");
      }
      catch(Exception e)
      { }
      System.out.println("---------");

      long m1 = ti.findfact(20);
      System.out.println(m1);
      System.out.println("-----------");

      long m2 = ti.findfact(58);
      System.out.println(m2);

   }
}
-------------------------------------------------------------------------------
=====================Spring transactions(using spring DAO)=====================
App 65)>>>>>>>>>>>>>>>>Programmatic Spring transactions(local,flat Tx)<<<<<<<<<<<<<<<<
==============================================================================
----------------------------Demo.java-----------------------
public interface Demo
{
    public void mymethod();
}
----------------------------DemoBean.java-------------------
import org.springframework.transaction.support.*;
import org.springframework.transaction.*;
import org.springframework.jdbc.core.*;

public class DemoBean implements Demo
{
  TransactionTemplate tt;
  JdbcTemplate jt;
  public void setJt(JdbcTemplate jt)
   {
      this.jt=jt;
   }
  public void setTt(TransactionTemplate tt)
   {
      this.tt=tt;
   }
  public void mymethod()
   {
```

giving instructions to generate proxy classes at compile time through annotations.

jar utility

same as app6

Spring Interface

declaration of business method

Spring bean class.

JPG to PDF Converter For Mac - Unregis

here execute method is called having an anymymous inner class object that implements TransactionCallback interface. *1

```
tt.execute(new TransactionCallback()
{
    public Object doInTransaction(TransactionStatus ts)
    {
    try
    {
    int r1=jt.update("insert into dept values(14,'mydept12','hyd12')");
    int r2=jt.update("update emp set sal=10000 where empno=7499");

    if(r1==0 || r2==0)
    {
    ts.setRollbackOnly();
    System.out.println("Tx is rolledback");
    }
    else
    System.out.println("Tx is committed");
    }
    catch(Exception e)
    {
    ts.setRollbackOnly();
    System.out.println("Tx is rolledback");
    }
    return null;
  }//doInTransaction
});//anonymous inner class
}//my method
}//class
```

*2 → It marks the transaction to rollback that means at the end of business method the TransactionTemplate class will rollback the transaction.

normally fixed b/c engg way

If the logic of doInTransaction() method executes without touching ts.setRollbackOnly() method; then the transaction will be committed by TransactionTemplate class at the end of the business method

If the same code executes with touching ts.setRollbackOnly() then the txn will be rolledback at the end of the bus. method

spring configuration file

----------demo.xml----------

```
<?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="drds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName"><value>oracle.jdbc.driver.OracleDriver</value> </property>
  <property name="url"><value>jdbc:oracle:thin:@localhost:1521:orcl</value></property>
   <property name="username"> <value>scott</value></property>
   <property name="password"> <value>tiger</value></property>
  </bean>

  <bean id="dst" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource"><ref bean="drds"/></property>
  </bean>

  <bean id="tt1" class="org.springframework.transaction.support.TransactionTemplate">
    <property name="transactionManager"><ref bean="dst"/></property>
  </bean>

  <bean id="jt1" class="org.springframework.jdbc.core.JdbcTemplate">
    <property name="dataSource"><ref bean="drds"/> </property>
  </bean>

  <bean id="db" class="DemoBean">
    <property name="tt"> <ref bean="tt1"/> </property>
    <property name="jt"><ref bean="jt1"/> </property>
  </bean>
  </beans>
```

gives DataSource object [recommended to use either C3P0 or apache dbcp]

ref 820

ref 827

ref 820

ref 831

ref 88

client application

----------ClientApp.java----------

```
import org.springframework.context.support.*;
public class ClientApp
{
```

Bean class injected with two

JPG to PDF Converter For Mac - Unregis

jar file in class path
spring.jar, commons-logging.jar

```
public static void main(String args[ ])
 {
      FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext("demo.xml");
      Demo  beanref =(Demo)ctx.getBean("db");
      beanref.mymethod();
 }
}
-----------------------------------------------------------------------------
=============================================================================
App 66) Declarative Spring transactions(local,flat Tx)
=============================================================================
---------------------Demo.java---------------------
public interface Demo
{
  public void bm1();
}
---------------------------DemoBean.java-------------------
import org.springframework.jdbc.core.*;
public class DemoBean implements Demo
{
  JdbcTemplate jt;
   public void setJt(JdbcTemplate jt)
   {
       this.jt=jt;
   }

   public void bm1(){
     int r2=jt.update("insert into dept values(32,'abc','hyd')");
     int r1=jt.update("update emp set sal=5000 where empno=74001");

     if(r1==0 || r2==0)
      {
        System.out.println("Tx is rolledback");
        throw new java.lang.RuntimeException();
      }
      else
       System.out.println("Tx is commited");
   }
}
-----------------------------------------------demo.xml-----------------------------
<?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="drds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
   <property name="driverClassName"><value>oracle.jdbc.driver.OracleDriver</value></property>
    <property name="url"><value>jdbc:oracle:thin:@localhost:1521:orcl</value></property>
    <property name="username"><value>scott</value></property>
    <property name="password"><value>tiger</value></property>
   </bean>

   <bean id="dts" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource"><ref bean="drds"/></property>
   </bean>


    <bean id="jt1" class="org.springframework.jdbc.core.JdbcTemplate">
      <property name="dataSource"><ref bean="drds"/></property>
    </bean>

    <bean id="db" class="DemoBean">
```


JPG to PDF Converter For Mac - Unregis

```
    <property name="jt"><ref bean="jt1"/></property>
  </bean>

  <bean id="tas"
  class="org.springframework.transaction.interceptor.NameMatchTransactionAttributeSource">
    <property name="properties">
      <props>
        <prop key="bm1">PROPAGATION_REQUIRED</prop>
      </props>
    </property>
  </bean>

  <bean id="tfb"
  class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
    <property name="target">><ref bean="db"/></property>
    <property name="transactionManager"><ref bean="dts"/></property>
    <property name="transactionAttributeSource"><ref bean="tas"/></property>
  </bean>
</beans>
-----------------------------------------ClientApp.java-----------------------------------------
import org.springframework.context.support.*;
public class ClientApp
{
  public static void main(String args[ ]) throws Exception
  {
      FileSystemXmlApplicationContext ctx=new FileSystemXmlApplicationContext("demo.xml");
      Demo beanref=(Demo)ctx.getBean("tfb");
      beanref.bm1();
  }
}
-----------------------------------------
=========================================
App 67) Declarative Transactions(Flat,Local) Using Annotations
=========================================
----------------------Demo.java-----------------------
public  interface  Demo
{
      public void bm1()throws RuntimeException;
}
-------------------------DemoBean.java-------------------
import  org.springframework.transaction.annotation.*;
import org.springframework.jdbc.core.*;
public class DemoBean  implements Demo{
 JdbcTemplate jt;
 public void setJt(JdbcTemplate jt) {
      this.jt = jt;
 }

 @Transactional(propagation=Propagation.REQUIRED)
 public void bm1()throws RuntimeException
 {
      int res1=jt.update("update emp set sal=9999 where empno=74991");
      int res2=jt.update("update dept set dname='payroll1'  where deptno=31");

      if(res1==0 || res2==0)
      {
        System.out.println("Tx rolled back");
        throw new RuntimeException();
      }
      else
      {
```


JPG to PDF Converter For Mac - Unregis

```
    System.out.println("Tx Committed");
    }
  }//bm1
}//class
-----------------------------------spcfg.xml----------------------------
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:aop="http://www.springframework.org/schema/aop"
xmlns:tx="http://www.springframework.org/schema/tx"
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.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd">

 <bean id="drds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
   <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
   <property name="url" value="jdbc:oracle:thin:@localhost:1521:orcl"/>
   <property name="username"  value="scott"/>
   <property name="password"  value="tiger"/>
 </bean>

 <bean  id="txmgr"
 class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
   <property name="dataSource"  ref="drds"/>
 </bean>

  <bean id="template" class="org.springframework.jdbc.core.JdbcTemplate">
   <constructor-arg  ref="drds"/>
 </bean>

 <bean id="db"  class="DemoBean">
    <property name="jt" ref="template"/>
</bean>

  <tx:annotation-driven   transaction-manager="txmgr"/>
 </beans>
 -----------------------------------ClientApp.java-----------------------------
import org.springframework.context.*;
import org.springframework.context.support.*;
public class ClientApp {
  public static void main(String args[])throws Exception
  {
      ApplicationContext ctx=new ClassPathXmlApplicationContext("spcfg.xml");
      Demo proxy =(Demo)ctx.getBean("db");
      proxy.bm1();
  }
}

=================================================================================
App 68)>>>>>>Spring programatic transactions(using Hibernate Transaction
manager)(Flat,Local)<<<<<<<<<<<<<<<<<<<<<<
=================================================================================
--------------------Demo.java-----------------------
public interface Demo
{
      public boolean transferMoney(int srcid,int destid,float amt);
}
```


JPG to PDF Converter For Mac - Unregis

```
----------------------DemoBean.java----------------------------
import org.springframework.transaction.support.*;
import org.springframework.transaction.*;
import org.springframework.orm.hibernate3.*;

public class DemoBean implements Demo
{
TransactionTemplate tt;
HibernateTemplate ht;
public void setHt(HibernateTemplate ht)
{
      this.ht=ht;
}
public void setTt(TransactionTemplate tt)
{
  this.tt=tt;
}

public boolean transferMoney(final int srcid,final int destid,final float amt)
{

  Boolean result=(Boolean)tt.execute(new TransactionCallback()
      {
      public Object doInTransaction(TransactionStatus ts)
      {
      boolean status=false;

    try
    {
      //transferMoney Logic
       int r1=ht.bulkUpdate("update Account a1 set a1.balance=a1.balance-? where a1.acid=?",
                            new Object[]{new Float(amt),new Integer(srcid)});

       int r2=ht.bulkUpdate("update Account a1 set a1.balance=a1.balance+? where a1.acid=?",
                            new Object[]{new Float(amt),new Integer(destid)});

      if(r1==0 || r2==0)
      {
       status=false;
       ts.setRollbackOnly();
       System.out.println("Tx is rolledback");
      }
      else
      {
       System.out.println("Tx is committed");
       status=true;
      }
    }//try
    catch(Exception e)
    {
       status=false;
       ts.setRollbackOnly();
       System.out.println("Tx is rolledback");
    }
       return new Boolean(status);
     }//doInTransaction()
  });

  return result.booleanValue();
 }//transferMoney()
```

HQL Query

HQL Query

JPG to PDF Converter For Mac - Unregis

```
}//class
----------------------------------------Account.java (HB pojo class)----------------------------------
public class Account
{
 int acid;
 String holdername;
 float balance;

 public void setAcid(int acid){this.acid=acid;}
 public int getAcid(){    return acid;}
 public void setHoldername(String holdername){this.holdername=holdername;}
 public String getHoldername(){  return holdername;}
 public void setBalance(float balance){     this.balance=balance;}
 public float getBalance(){return balance;}
}
-------------------------------------Account.hbm.xml----------------------------
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
 <class name="Account"  table="HB_Account">
   <id name="acid" column="ACID" >
     <generator class="increment"/>
   </id>
   <property name="holdername" column="HOLDERNAME"/>
   <property name="balance" column="BALANCE"/>
 </class>
</hibernate-mapping>
-------------------------------mycfg.xml----------------------
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <property
    name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:orcl</property>
    <property name="hibernate.connection.username">scott</property>
    <property name="hibernate.connection.password">tiger</property>
    <property name="hibernate.dialect">org.hibernate.dialect.OracleDialect</property>
    <property name="show_sql">true</property>
    <mapping resource="Account.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
-------------------------------------SpringCfg.xml----------------------------------------
<!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
 <bean id="mySessionFactory"
 class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
   <property name="configLocation"><value>classpath:mycfg.xml</value></property>
 </bean>

 <bean id="template" class="org.springframework.orm.hibernate3.HibernateTemplate">
   <constructor-arg><ref bean="mySessionFactory"/></constructor-arg>
 </bean>
```

gives Hibernate SessionFactory object

specifying the name of Hibernate configuration file

ref 1139

JPG to PDF Converter For Mac - Unregis

```
<bean id="hbt" class="org.springframework.orm.hibernate3.HibernateTransactionManager">
   <property name="sessionFactory" ref="mySessionFactory"/>
</bean>

<bean id="tt1" class="org.springframework.transaction.support.TransactionTemplate">
   <property name="transactionManager"><ref bean="hbt"/></property>
</bean>
<bean id="db" class="DemoBean">
   <property name="ht"><ref bean="template"/></property>
   <property name="tt"><ref bean="tt1"/></property>
</bean>
</beans>
```

ref 1139 → To perform Transaction management on Hibernate code

ref 1147

1157 1143 → our spring bean where hibernate template, transaction template class objects are injected

```
----------------------------------------ClientApp.java----------------------------
import org.springframework.context.support.*;
public class ClientApp
{
 public static void main(String args[ ])
 {
     FileSystemXmlApplicationContext ctx=new
     FileSystemXmlApplicationContext("SpringCfg.xml");
     Demo  bean =(Demo)ctx.getBean("db");
     boolean res=bean.transferMoney(102,101,3000);
     if(res)
      System.out.println("Money Transfered");
     else
      System.out.println("Money not Transfered");
 }
}
```

→ calling the bean's method

```
==================================================
App69)>>>>>>Spring Declarative transactions(using Hibernate Transaction
manager)(Flat,Local)<<<<<<<<<<<<<
==================================================
--------------------------------Demo.java-------------------------
public interface Demo
{
     public boolean transferMoney(int srcid,int destid,float amt)throws RuntimeException;
}
-----------------------DemoBean.java-----------------
import org.springframework.transaction.support.*;
import org.springframework.transaction.*;
import org.springframework.orm.hibernate3.*;

public class DemoBean implements Demo
{
 HibernateTemplate ht;

 public void setHt(HibernateTemplate ht){this.ht=ht;}

 public boolean transferMoney( int srcid, int destid, float amt)throws RuntimeException
 {
   boolean status=false;

   int r1=ht.bulkUpdate("update Account a1 set a1.balance=a1.balance-? where a1.acid=?",
                        new Object[]{new Float(amt),new Integer(srcid)});
   int r2=ht.bulkUpdate("update Account a1 set a1.balance=a1.balance+? where a1.acid=?",
                        new Object[]{new Float(amt),new Integer(destid)});

    if(r1==0 || r2==0)
    {
     status=false;
    }
```

→ For withdrawal operation

→ For deposit operation.

JPG to PDF Converter For Mac - Unregis

```
        else
        {
         System.out.println("Tx committed");
         status=true;
        }
        if(status==false){
         throw new RuntimeException();
        }
        return status;
    }//transferMoney()
}//class
----------------------------Account.java----------------------
public class Account
{
  int acid;
  String holdername;
  float balance;

  public void setAcid(int acid) {this.acid=acid;}
  public int getAcid(){    return acid;}
  public void setHoldername(String holdername){    this.holdername=holdername;}
  public String getHoldername(){ return holdername;}
  public void setBalance(float balance){this.balance=balance;}
  public float getBalance(){   return balance;}
}
-------------------------------------Account.hbm.xml-------------------------
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class name="Account"  table="HB_Account">
   <id name="acid" column="ACID" >
     <generator class="increment"/>
   </id>
   <property name="holdername"/>
   <property name="balance"/>
  </class>
</hibernate-mapping>
-----------------------------------mycfg.xml----------------------
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
   <property
   name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
   <property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:orcl</property>
   <property name="hibernate.connection.username">scott</property>
   <property name="hibernate.connection.password">tiger</property>
   <property name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
   <property name="show_sql">true</property>
   <mapping resource="Account.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
-------------------------------SpringCfg.xml-----------------------------
<!DOCTYPE beans PUBLIC  "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
```

make the transaction manager to rollback the txn

JPG to PDF Converter For Mac - Unregis

```
<bean id="mySessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
  <property name="configLocation"><value>classpath:mycfg.xml</value></property>
</bean>

<bean id="template" class="org.springframework.orm.hibernate3.HibernateTemplate">
   <constructor-arg><ref bean="mySessionFactory"/></constructor-arg>
</bean>

<bean id="hbt" class="org.springframework.orm.hibernate3.HibernateTransactionManager">
   <property name="sessionFactory" ref="mySessionFactory"/>
</bean>

<bean id="db" class="DemoBean">
   <property name="ht"><ref bean="template"/></property>
</bean>

 <bean id="tas"
 class="org.springframework.transaction.interceptor.NameMatchTransactionAttributeSource">
  <property name="properties">
    <props>
      <prop key="transferMoney">PROPAGATION_REQUIRED</prop>
    </props>
  </property>
  </bean>

 <bean id="tfb"
 class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
   <property name="target"><ref bean="db"/></property>
   <property name="transactionManager"><ref bean="hbt"/></property>
   <property name="transactionAttributeSource"><ref bean="tas"/></property>
  </bean>
</beans>
```

local sessionfactory configuration

gives hibernate session factory object

ref 1266

ref 1266

to perform transaction management on hibernate persistance logic

ref 1270

to apply Transaction attributes on business methods

The transaction att required is applied on the business method transfer money()

ref 1274

use proxy object having transaction management service.

ref 1278 (1282) (or spring bean)

```
----------------------------------ClientApp.java----------------------------------
import org.springframework.context.support.*;
public class ClientApp
{
 public static void main(String args[ ])
 {
     FileSystemXmlApplicationContext ctx=new
     FileSystemXmlApplicationContext("SpringCfg.xml");
     Demo  bean =(Demo)ctx.getBean("tfb");
     boolean res=bean.transferMoney(102,101,6000);
     if(res)
      System.out.println("Money Transfered");
     else
      System.out.println("Money not Transfered");
 }
}
```

ref 1290

This business method is called without transaction in client application since required transaction is applied the business method always runs with new transaction

```
==========================Spring Application on Global /Distributed Transactions <<<<<<<<<<<<<<
App 70)>>>
```

Account1 → oracle DBS/w DB table of Account2 → MySQL DB S/w DB table of


```
public class Account implements AccountInter
{
 private JdbcTemplate jt1;
 private JdbcTemplate jt2;
 public void setJt1(JdbcTemplate jt1){this.jt1=jt1;}
 public void setJt2(JdbcTemplate jt2){this.jt2=jt2;}

 @Transactional
  public void transferMoney(int acc1,int acc2,int amount)throws RuntimeException
  {
     int s1=jt1.queryForInt("select bal from Account1 where accno=?",new Object[]{acc1});
     int s2 = s1-amount;

     int s3=jt2.queryForInt("select bal from Account2 where accno=?",new Object[]{acc2});
     int s4 = s3+amount;

     int r1=jt1.update("update Account1 set bal=? where accno=?",new Object[]{s2,acc1});
     int r2=jt2.update("update Account2 set bal=? where accno=?",new Object[]{s4,acc2});
     if(r1==0 || r2==0)
     {
         System.out.println("Tx rolled back (Money not transffered");
         throw new RuntimeException();
     }//if
     else
     {
      System.out.println("Tx Committed(Money transffered");
     }
  }//method
}//class
----------spring.xml----------
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:tx="http://www.springframework.org/schema/tx"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">
<bean id="dataSourceA" class="com.atomikos.jdbc.AtomikosDataSourceBean"
   init-method="init" destroy-method="close">
 <property name="uniqueResourceName"><value>XADBMSA</value></property>
 <property
 name="xaDataSourceClassName"><value>oracle.jdbc.xa.client.OracleXADataSource</value></pr
 operty>
 <property name="xaProperties">
  <props>
   <prop key="databaseName">ORCL</prop>
    <prop key="user">scott</prop>
     <prop key="password">tiger</prop>
     <prop key="URL">jdbc:oracle:thin:@localhost:1521:orcl</prop>
  </props>
  </property>
  <property name="poolSize">><value>10</value></property>
</bean>

<bean id="dataSourceB" class="com.atomikos.jdbc.AtomikosDataSourceBean"
       init-method="init" destroy-method="close">
  <property name="uniqueResourceName">><value>XADBMSB</value></property>
  <property
  name="xaDataSourceClassName">><value>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource<
  /value>
```

two JdbcTemplate class objects are taken pointing to two different database softwares

enables transaction management on the business method

for withdrawal operation

for deposit operation

updating withdrawal operation

updating deposit operation

DB table in mysql [in mysql logical DB]

accno	tablename	bal
102	orapl	1000

gives xaDataSource object pointing to one database S/w

instruction to give xaDataSource object pointing to oracle

special class to give xaDataSource object pointing to oracle

perform global Transaction management

JPG to PDF Converter For Mac - Unregis

```
1380  </property>
1381
1382  <property name="xaProperties">
1383   <props>
1384     <prop key="databaseName">mydb1</prop>
1385     <prop key="user">root1</prop>
1386      <prop key="password">root1</prop>
1387    </props>
1388   </property>
1389  <property name="poolSize"><value>10</value></property>
1390  </bean>
1391
1392  <!-- Construct Atomikos UserTransactionManager, needed to configure Spring -->
1393  <bean id="atomikosTransactionManager"
      class="com.atomikos.icatch.jta.UserTransactionManager"
1394          init-method="init" destroy-method="close">
1395  <!-- when close is called, should we force transactions to terminate or not? -->
1396    <property name="forceShutdown">><value>true</value></property>
1397  </bean>
1398
1399  <!-- Also use Atomikos UserTransactionImp, needed to configure Spring -->
1400  <bean id="atomikosUserTransaction" class="com.atomikos.icatch.jta.UserTransactionImp">
1401    <property name="transactionTimeout">><value>300</value></property>
1402  </bean>
1403
1404  <!-- Configure the Spring framework to use JTA transactions from Atomikos -->
1405  <bean id="tm" class="org.springframework.transaction.jta.JtaTransactionManager">
1406     <property name="transactionManager"> <ref
       bean="atomikosTransactionManager"/></property>
1407      <property name="userTransaction"><ref bean="atomikosUserTransaction"/></property>
1408  </bean>
1409
1410  <tx:annotation-driven transaction-manager="tm"/>
1411
1412  <bean id="jt1" class="org.springframework.jdbc.core.JdbcTemplate">
1413    <constructor-arg ref="dataSourceA"/>
1414  </bean>
1415
1416  <bean id="jt2" class="org.springframework.jdbc.core.JdbcTemplate">
1417    <constructor-arg ref="dataSourceB"/>
1418  </bean>
1419
1420  <bean id="ac1" class="Account">
1421    <property name="jt1" ref="jt1"/>
1422    <property name="jt2" ref="jt2"/>
1423  </bean>
1424
1425  </beans>
1426  ----------------------------Client.java----------------------------
1427  import org.springframework.context.*;
1428  import org.springframework.context.support.*;
1429  class Client
1430  { public static void main(String[] args)
1431    {
1432        ApplicationContext ctx=new ClassPathXmlApplicationContext("spring.xml");
1433        AccountInter ac =(AccountInter)ctx.getBean("ac1");
1434        ac.transferMoney(101,102,1000);
1435    }
1436  }
1437
1438
```

gives XADataSource object pointing to MYSQL

logical database name.

The transaction will be aborted automatically if it is not committed or rolled back once it is started and continuously running more than 300 secs

Transaction manager supporting Distributed transaction

to specify additional information related to Transaction Management

ref 1393

ref 1400

To perform transaction Management by using the above atomikos supplied transaction service

ref 1361

ref 1378

To perform transaction Management through Annotations.

JdbcTemplate object pointing to oracle DBS/W

gives JdbcTemplate class obj pointing to mysql DBS S/W

ref 1420

our spring bean configuration.

calling business method

JPG to PDF Converter For Mac - Unregis

```
==========================================================================
App 71)  >>>>>>>>>>>>>>>>>>Application  on SpringSecurity Plugin (Acegi)<<<<<<<<<<<<<<<<
==========================================================================
----------------------------index.jsp----------------------------
Anyone can view this page.
<p><a href="secure/Admin.jsp">Admin Page</a></p>
<p><a href="secure/Faculty.jsp">Faculty Page</a></p>
----------------------------web.xml----------------------------
<web-app>
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
  <context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/mySecurityConfig.xml</param-value>
  </context-param>
  <welcome-file-list>
      <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
  <filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
  </filter>

  <filter-mapping>
   <filter-name>springSecurityFilterChain</filter-name>
   <url-pattern>/*</url-pattern>
  </filter-mapping>
</web-app>
----------------------------Admin.jsp----------------------------
<h1>Admin Page</h1>
 Welcome Mr:<%=request.getUserPrincipal().getName()%><br><br><br>
 Wanna goto Faculty Page <a href="Faculty.jsp">click here</a>
 <p><a href="../j_spring_security_logout">Logout</a>
----------------------------Faculty.jsp----------------------------
<h1> Faculty Page </h1>
 Welcome Mr:<%=request.getUserPrincipal().getName()%><br><br><br>
 Wanna goto Admin Page <a href="Admin.jsp">click here</a>
<p><a href="../j_spring_security_logout">Logout</a>
----------------------------mySecurityConfig.xml----------------------------
<beans:beans xmlns="http://www.springframework.org/schema/security"
    xmlns:beans="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-3.0.xsd
   http://www.springframework.org/schema/security
   http://www.springframework.org/schema/security/spring-security-3.0.3.xsd">

  <http use-expressions="true">
*1 <intercept-url pattern="/index.jsp" access="permitAll" />
*2 <intercept-url pattern="/secure/Admin.jsp" access="hasRole('admin')" />
*3 <intercept-url pattern="/secure/Faculty.jsp" access="hasRole('faculty')" />
     <form-login />
     <logout />
     <remember-me />
     <session-management invalid-session-url="/logout.jsp">
        <concurrency-control max-sessions="3" error-if-maximum-exceeded="true" />
     </session-management>
     </http>

  <authentication-manager>
```

DD file

servletContextListener to activate the spring container during server startup

fixed param name

allows to configure security roles.

servlet Filter acting as the controller of the web app handling the ability to trap all requests and responses

resource

use the currently logged in username

the fixed hard word

resource

supporting xml file to security plugin

allows regular expressions

launches the form page dynamically asking username and password

makes application to remember the username during session

*1 → Any user can access and use index.jsp

*2 → Admin.jsp is accessible only to admin role users

*3 → Faculty.jsp is accessible only to Faculty role users

JPG to PDF Converter For Mac - Unregis

→ defining users specifying roles with them.

```
1500      <authentication-provider>
1501        <user-service>
1502          <user name="durga" password="soft" authorities="admin,faculty" />
1503          <user name="nataraz" password="nataraz" authorities="faculty" />
1504        </user-service>
1505      </authentication-provider>
1506    </authentication-manager>
1507  </beans:beans>
1508  ------------------------logout.jsp------------------
1509  You have successfully LoggedOut<br>
1510  <a href="index.jsp">start again</a>
1511  ------------------------------------------------------------------------
1512
```

→ page that executes when session is expiring or terminated

*4 → allows only 3 clients at a time to use this application

*5 → It is the url of spring security plugin supplied web resource who takes care of logout operations.

→ allows each user to login maximum of 3 times

SpringSecurityApp
- → WEB-INF
  - → secure
    - → Admin.jsp
    - → Faculty.jsp
  - → Index.jsp
  - → logout.jsp
  - → classes
  - → lib
    - → *.jar
  - → web.xml
  - → mySecurityConfig.xml

jars ability in WEB-INF/lib:-

spring-security-config-3.0.5-RELEASE
spring-security-core-3.0.5-RELEASE
spring-security-web-3.0.5-Release
→ commons-logging
- spring-aop-3.0.3-release
- spring-aym-3.0.3-release
- spring-beans-3.0.3-release
- spring-context
- spring-context-support
- spring-core-3.0.3
- spring-expression
- spring-EX-web-
- spring-jdbc
- spring-

→ gather from spring sec. (by extracting spring-security plugin software [spring-security-3.0.5.release.zip file])

JPG to PDF Converter For Mac - Unregis

What's new in Spring 2.0 and 2.5?

2.1. Introduction

If you have been using the Spring Framework for some time, you will be aware that Spring has undergone two major revisions: Spring 2.0, released in October 2006, and Spring 2.5, released in November 2007.

Java SE and Java EE Support

The Spring Framework continues to be compatible with all versions of Java since (and including) Java 1.4.2. This means that Java 1.4.2, Java 5 and Java 6 are supported, although some advanced functionality of the Spring Framework will not be available to you if you are committed to using Java 1.4.2. Spring 2.5 introduces dedicated support for Java 6, after Spring 2.0's in-depth support for Java 5 throughout the framework.

Furthermore, Spring remains compatible with J2EE 1.3 and higher, while at the same time introducing dedicated support for Java EE 5. This means that Spring can be consistently used on application servers such as BEA WebLogic 8.1, 9.0, 9.2 and 10, IBM WebSphere 5.1, 6.0 and 6.1, Oracle OC4J 10.1.3 and 11, JBoss 3.2, 4.0 and 4.2, as well as Tomcat 4.1, 5.0, 5.5 and 6.0, Jetty 4.2, 5.1 and 6.1, Resin 2.1, 3.0 and 3.1 and GlassFish V1 and V2.

This chapter is a guide to the new and improved features of Spring 2.0 and 2.5. It is intended to provide a high-level summary so that seasoned Spring architects and developers can become immediately familiar with the new Spring 2.x functionality. For more in-depth information on the features, please refer to the corresponding sections hyperlinked from within this chapter.

2.2. The Inversion of Control (IoC) container

One of the areas that contains a considerable number of 2.0 and 2.5 improvements is Spring's IoC container.

2.2.1. New bean scopes

Previous versions of Spring had IoC container level support for exactly two distinct bean scopes (singleton and prototype). Spring 2.0 improves on this by not only providing a number of additional scopes depending on the environment in which Spring is being deployed (for example, request and session scoped beans in a web environment), but also by providing integration points so that Spring users can create their own scopes.

It should be noted that although the underlying (and internal) implementation for singleton- and prototype-scoped beans has been changed, this change is totally transparent to the end user... no existing configuration needs to change, and no existing configuration will break.

2.2.2. Easier XML configuration

Spring XML configuration is now even easier, thanks to the advent of the new XML configuration syntax based on XML Schema. On a related note, there is a new, updated DTD for Spring 2.0 that you may wish to reference if you cannot take advantage of the XML Schema-based configuration. The DOCTYPE declaration is included below for your convenience, but the interested reader should definitely read the 'spring-beans-2.0.dtd' DTD included in the 'dist/resources' directory of the Spring 2.5 distribution.

```
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
                "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
```

2.2.3. Extensible XML authoring

Not only is XML configuration easier to write, it is now also extensible.

What 'extensible' means in this context is that you, as an application developer, or (more likely) as a third party framework or product vendor, can write custom tags that other developers can

JPG to PDF Converter For Mac - Unregis

then plug into their own Spring configuration files. This allows you to have your own domain specific language (the term is used loosely here) of sorts be reflected in the specific configuration of your own components.

Implementing custom Spring tags may not be of interest to every single application developer or enterprise architect using Spring in their own projects. We expect third-party vendors to be highly interested in developing custom configuration tags for use in Spring configuration files.

2.2.4. Annotation-driven configuration

Spring 2.0 introduced support for various annotations for configuration purposes, such as @Transactional, @Required and @PersistenceContext /@PersistenceUnit.

Spring 2.5 introduces support for a complete set of configuration annotations: @Autowired in combination with support for the JSR-250 annotations @Resource, @PostConstruct and @PreDestroy .

2.2.5. Autodetecting components in the classpath

Spring 2.5 introduces support component scanning: autodetecting annotated components in the classpath. Typically, such component classes will be annotated with stereotypes such as @Component, @Repository, @Service, @Controller. Depending on the application context configuration, such component classes will be autodetected and turned into Spring bean definitions, not requiring explicit configuration for each such bean.

2.3. Aspect Oriented Programming (AOP)

Spring 2.0 has a much improved AOP offering. The Spring AOP framework itself is markedly easier to configure in XML, and significantly less verbose as a result; and Spring 2.0 integrates with the AspectJ pointcut language and @AspectJ aspect declaration style. The chapter entitled

2.3.1. Easier AOP XML configuration

Spring 2.0 introduces new schema support for defining aspects backed by regular Java objects. This support takes advantage of the AspectJ pointcut language and offers fully typed advice (i.e. no more casting and Object[] argument manipulation).

2.3.2. Support for @AspectJ aspects

Spring 2.0 also supports aspects defined using the @AspectJ annotations. These aspects can be shared between AspectJ and Spring AOP, and require (honestly!) only some simple configuration.

2.3.3. Support for bean name pointcut element

Spring 2.5 introduces support for the bean(...) pointcut element, matching specific named beans according to Spring-defined bean names.

2.3.4. Support for AspectJ load-time weaving

Spring 2.5 introduces explicit support AspectJ load-time weaving, as alternative to the proxy-based AOP framework. The new context:load-time-weaver configuration element automatically activates AspectJ aspects as defined in AspectJ's META-INF/aop.xml descriptor, applying them to the current application context through registering a transformer with the underlying ClassLoader. Note that this only works in environments with class transformation support.

2.4. The Middle Tier

2.4.1. Easier configuration of declarative transactions in XML

The way that transactions are configured in Spring 2.0 has been changed significantly. The previous 1.2.x style of configuration continues to be valid (and supported), but the new style is markedly less verbose and is the recommended style. Spring 2.0 also ships with an AspectJ aspects library that you can use to make pretty much any object transactional - even objects not created by the Spring IoC container.

JPG to PDF Converter For Mac - Unregis

Spring 2.5 supports convenient annotation-driven transaction management in combination with load-time weaving, through the use of context:load-time-weaver in combination with tx:annotation-driven mode="aspectj".

2.4.2. Full WebSphere transaction management support

Spring 2.5 explicitly supports IBM's WebSphere Application Server, in particular with respect to WebSphere's transaction manager. Transaction suspension is now fully supported through the use of WebSphere's new UOWManager API, which is available on WAS 6.0.2.19+ and 6.0.1.9+.

So if you run a Spring-based application on the WebSphere Application Server, we highly recommend to use Spring 2.5's WebSphereUowTransactionManager as your PlatformTransactionManager of choice. This is also IBM's official recommendation.

For automatic detection of the underlying JTA-based transaction platform, consider the use of Spring 2.5's new tx:jta-transaction-manager configuration element. This will autodetect BEA WebLogic and IBM WebSphere, registering the appropriate PlatformTransactionManager.

2.4.3. JPA

Spring 2.0 ships with a JPA abstraction layer that is similar in intent to Spring's JDBC abstraction layer in terms of scope and general usage patterns.

Spring 2.5 upgrades its OpenJPA support to OpenJPA 1.0, with support for advanced features such as savepoints.

2.4.4. Asynchronous JMS

Prior to Spring 2.0, Spring's JMS offering was limited to sending messages and the *synchronous* receiving of messages. This functionality (encapsulated in the JmsTemplate class) is great, but it doesn't address the requirement for the *asynchronous* receiving of messages.

Spring 2.0 now ships with full support for the reception of messages in an asynchronous fashion..

As of Spring 2.5, the JCA style of setting up asynchronous message listeners is supported as well, through the GenericMessageEndpointManager facility. This is an alternative to the standard JMS listener facility, allowing closer integration with message brokers such as ActiveMQ and JORAM.

Spring 2.5 also introduces an XML namespace for simplifying JMS configuration, offering concise configuration of a large numbers of listeners. This namespace supports both the standard JMS listener facility as well as the JCA setup style, with minimal changes in the configuration.

2.4.5. JDBC

There are some small (but nevertheless notable) new classes in the Spring Framework's JDBC support library. The first, NamedParameterJdbcTemplate, provides support for programming JDBC statements using named parameters (as opposed to programming JDBC statements using only classic placeholder ('?') arguments.

Another of the new classes, the SimpleJdbcTemplate, is aimed at making using the JdbcTemplate even easier to use when you are developing against Java 5+ (Tiger).

Spring 2.5 significantly extends the functionality of SimpleJdbcTemplate and introduces SimpleJdbcCall and SimpleJdbcInsert operation objects.

2.5. The Web Tier

The web tier support has been *substantially* improved and expanded in Spring 2.0, with annotation-based controllers introduced in Spring 2.5.

2.5.1. Sensible defaulting in Spring MVC

For a lot of projects, sticking to established conventions and having reasonable defaults is just what the projects need... this theme of convention-over-configuration now has explicit support in

JPG to PDF Converter For Mac - Unregis

Spring MVC. What this means is that if you establish a set of naming conventions for your Controllers and views, you can *substantially* cut down on the amount of XML configuration that is required to setup handler mappings, view resolvers, ModelAndView instances, etc. This is a great boon with regards to rapid prototyping, and can also lend a degree of (always good-to-have) consistency across a codebase.

2.5.2. Portlet framework

Spring 2.0 ships with a Portlet framework that is conceptually similar to the Spring MVC framework.

2.5.3. Annotation-based controllers

Spring 2.5 introduces an annotation-based programming model for MVC controllers, using annotations such as @RequestMapping, @RequestParam, @ModelAttribute, etc. This annotation support is available for both Servlet MVC and Portlet MVC. Controllers implemented in this style do not have to extend specific base classes or implement specific interfaces. Furthermore, they do not usually have direct dependencies on Servlet or Portlet API's, although they can easily get access to Servlet or Portlet facilities if desired.

2.5.4. A form tag library for Spring MVC

A rich JSP tag library for Spring MVC was *the* JIRA issue that garnered the most votes from Spring users (by a wide margin).

Spring 2.0 ships with a full featured JSP tag library that makes the job of authoring JSP pages much easier when using Spring MVC; the Spring team is confident it will satisfy all of those developers who voted for the issue on JIRA.

2.5.5. Tiles 2 support

Spring 2.5 ships support for Tiles 2, the next generation of the popular Tiles templating framework. This supersedes Spring's former support for Tiles 1, as included in Struts 1.x.

2.5.6. JSF 1.2 support

Spring 2.5 supports JSF 1.2, providing a JSF 1.2 variant of Spring's DelegatingVariableResolver in the form of the new SpringBeanFacesELResolver.

2.5.7. JAX-WS support

Spring 2.5 fully supports JAX-WS 2.0/2.1, as included in Java 6 and Java EE 5. JAX-WS is the successor of JAX-RPC, allowing access to WSDL/SOAP-based web services as well as JAX-WS style exposure of web services.

2.6. Everything else

This final section outlines all of the other new and improved Spring 2.0/2.5 features and functionality.

2.6.1. Dynamic language support

Spring 2.0 introduced support for beans written in languages other than Java, with the currently supported dynamic languages being JRuby, Groovy and BeanShell..

Spring 2.5 refines the dynamic languages support with autowiring and support for the recently released JRuby 1.0.

2.6.2. Enhanced testing support

Spring 2.5 introduces the *Spring TestContext Framework* which provides annotation-driven unit and integration testing support that is agnostic of the actual testing framework in use. The same techniques and annotation-based configuration used in, for example, a JUnit 3.8 environment can also be applied to tests written with JUnit 4.4, TestNG, etc.

JPG to PDF Converter For Mac - Unregis

In addition to providing generic and extensible testing infrastructure, the *Spring TestContext Framework* provides out-of-the-box support for Spring-specific integration testing functionality such as context management and caching, dependency injection of test fixtures, and transactional test management with default rollback semantics.

2.6.3. JMX support

The Spring Framework 2.0 has support for Notifications; it is also possible to exercise declarative control over the registration behavior of MBeans with an MBeanServer.

- Section 20.7, "Notifications"
- Section 20.2.5, "Controlling the registration behavior"

Furthermore, Spring 2.5 provides a context:mbean-export configuration element for convenient registration of annotated bean classes, detecting Spring's @ManagedResource annotation.

2.6.4. Deploying a Spring application context as JCA adapter

Spring 2.5 supports the deployment of a Spring application context as JCA resource adapter, packaged as a JCA RAR file. This allows headless application modules to be deployed into J2EE servers, getting access to all the server's infrastructure e.g. for executing scheduled tasks, listening for incoming messages, etc.

2.6.5. Task scheduling

Spring 2.0 offers an abstraction around the scheduling of tasks.

The TaskExecutor abstraction is used throughout the framework itself as well, e.g. for the asynchronous JMS support. In Spring 2.5, it is also used in the JCA environment support.

2.6.6. Java 5 (Tiger) support

Find below pointers to documentation describing some of the new Java 5 support in Spring 2.0 and 2.5.

- Section 3.11, "Annotation-based configuration"
- Section 25.3.1, "@Required"
- Section 9.5.6, "Using @Transactional"
- Section 11.2.3, "SimpleJdbcTemplate"
- Section 12.6, "JPA"
- Section 6.2, "@AspectJ support"
- Section 6.8.1, "Using AspectJ to dependency inject domain objects with Spring"

2.7. Migrating to Spring 2.5

This final section details issues that may arise during any migration from Spring 1.2/2.0 to Spring 2.5.

Upgrading to Spring 2.5 from a Spring 2.0.x application should simply be a matter of dropping the Spring 2.5 jar into the appropriate location in your application's directory structure. We highly recommend upgrading to Spring 2.5 from any Spring 2.0 application that runs on JDK 1.4.2 or higher, in particular when running on Java 5 or higher, leveraging the significant configuration conveniences and performance improvements that Spring 2.5 has to offer.

Whether an upgrade from Spring 1.2.x will be as seamless depends on how much of the Spring APIs you are using in your code. Spring 2.0 removed pretty much all of the classes and methods previously marked as deprecated in the Spring 1.2.x codebase, so if you have been using such classes and methods, you will of course have to use alternative classes and methods (some of which are summarized below).

JPG to PDF Converter For Mac - Unregis

With regards to configuration, Spring 1.2.x style XML configuration is 100%, satisfaction-guaranteed compatible with the Spring 2.5 library. Of course if you are still using the Spring 1.2.x DTD, then you won't be able to take advantage of some of the new Spring 2.0 functionality (such as scopes and easier AOP and transaction configuration), but nothing will blow up.

The suggested migration strategy is to drop in the Spring 2.5 jar(s) to benefit from the improved code present in the release (bug fixes, optimizations, etc.). You can then, on an incremental basis, choose to start using the new Spring 2.5 features and configuration. For example, you could choose to start configuring just your aspects in the new Spring 2 style; it is perfectly valid to have 90% of your configuration using the old-school Spring 1.2.x configuration (which references the 1.2.x DTD), and have the other 10% using the new Spring 2 configuration (which references the 2.0/2.5 DTD or XSD). Bear in mind that you are not forced to upgrade your XML configuration should you choose to drop in the Spring 2.5 libraries.

2.7.1. Changes

For a comprehensive list of changes, consult the 'changelog.txt' file that is located in the top level directory of the Spring Framework distribution.

2.7.1.1. Supported JDK versions

As of Spring 2.5, support for JDK 1.3 has been removed, following Sun's official deprecation of JDK 1.3 in late 2006. If you haven't done so already, upgrade to JDK 1.4.2 or higher.

If you need to stick with an application server that only supports JDK 1.3, such as WebSphere 4.0 or 5.0, we recommend using the Spring Framework version 2.0.7/2.0.8 which still supports JDK 1.3.

2.7.1.2. Jar packaging in Spring 2.5

As of Spring 2.5, Spring Web MVC is no longer part of the 'spring.jar' file. Spring MVC can be found in 'spring-webmvc.jar' and 'spring-webmvc-portlet.jar' in the lib/modules directory of the distribution. Furthermore, the Struts 1.x support has been factored out into 'spring-webmvc-struts.jar'.

Note: The commonly used Spring's DispatcherServlet is part of Spring's Web MVC framework. As a consequence, you need to add 'spring-webmvc.jar' (or 'spring-webmvc-portlet/struts.jar') to a 'spring.jar' scenario, even if you are just using DispatcherServlet for remoting purposes (e.g. exporting Hessian or HTTP invoker services).

Spring 2.0's 'spring-jmx.jar' and 'spring-remoting.jar' have been merged into Spring 2.5's 'spring-context.jar' (for the JMX and non-HTTP remoting support) and partly into 'spring-web.jar' (for the HTTP remoting support).

Spring 2.0's 'spring-support.jar' has been renamed to 'spring-context-support.jar', expressing the actual support relationship more closely. 'spring-portlet.jar' has been renamed to 'spring-webmvc-portlet.jar', since it is technically a submodule of Spring's Web MVC framework. Analogously, 'spring-struts.jar' has been renamed to 'spring-webmvc-struts.jar'.

Spring 2.0's 'spring-jdo.jar', 'spring-jpa.jar', 'spring-hibernate3.jar', 'spring-toplink.jar' and 'spring-ibatis.jar' have been combined into Spring 2.5's coarse-granular 'spring-orm.jar'.

Spring 2.5's 'spring-test.jar' supersedes the previous 'spring-mock.jar', indicating the stronger focus on the test context framework. Note that 'spring-test.jar' contains everything 'spring-mock.jar' contained in previous Spring versions; hence it can be used as a straightforward replacement for unit and integration testing purposes.

Spring 2.5's 'spring-tx.jar' supersedes the previous 'spring-dao.jar' and 'spring-jca.jar' files, indicating the stronger focus on the transaction framework.

Spring 2.5 ships its framework jars as OSGi-compliant bundles out of the box. This facilitates use of Spring in OSGi environments, not requiring custom packaging anymore.

JPG to PDF Converter For Mac - Unregis

2.7.1.3. XML configuration

Spring 2.0 ships with XSDs that describe Spring's XML metadata format in a much richer fashion than the DTD that shipped with previous versions. The old DTD is still fully supported, but if possible you are encouraged to reference the XSD files at the top of your bean definition files.

One thing that has changed in a (somewhat) breaking fashion is the way that bean scopes are defined. If you are using the Spring 1.2 DTD you can continue to use the 'singleton' attribute. You can however choose to reference the new Spring 2.0 DTD which does not permit the use of the 'singleton' attribute, but rather uses the 'scope' attribute to define the bean lifecycle scope.

2.7.1.4. Deprecated classes and methods

A number of classes and methods that previously were marked as @deprecated have been removed from the Spring 2.0 codebase. The Spring team decided that the 2.0 release marked a fresh start of sorts, and that any deprecated 'cruft' was better excised now instead of continuing to haunt the codebase for the foreseeable future.

As mentioned previously, for a comprehensive list of changes, consult the 'changelog.txt' file that is located in the top level directory of the Spring Framework distribution.

The following classes/interfaces have been removed as of Spring 2.0:

- ResultReader : Use the RowMapper interface instead.
- BeanFactoryBootstrap : Consider using a BeanFactoryLocator or a custom bootstrap class instead.

2.7.1.5. Apache OJB

As of Spring 2.0, support for Apache OJB was *totally removed* from the main Spring source tree. The Apache OJB integration library is still available, but can be found in it's new home in the Spring Modules project.

2.7.1.6. iBATIS

Please note that support for iBATIS SQL Maps 1.3 has been removed. If you haven't done so already, upgrade to iBATIS SQL Maps 2.3.

2.7.1.7. Hibernate

As of Spring 2.5, support for Hibernate 2.1 and Hibernate 3.0 has been removed. If you haven't done so already, upgrade to Hibernate 3.1 or higher.

If you need to stick with Hibernate 2.1 or 3.0 for the time being, we recommend to keep using the Spring Framework version 2.0.7/2.0.8 which still supports those versions of Hibernate.

2.7.1.8. JDO

As of Spring 2.5, support for JDO 1.0 has been removed. If you haven't done so already, upgrade to JDO 2.0 or higher.

If you need to stick with JDO 1.0 for the time being, we recommend to keep using the Spring Framework version 2.0.7/2.0.8 which still supports that version of JDO.

2.7.1.9. UrlFilenameViewController

Since Spring 2.0, the view name that is determined by the UrlFilenameViewController now takes into account the nested path of the request. This is a breaking change from the original contract of the UrlFilenameViewController, and means that if you are upgrading from Spring 1.x to Spring 2.x and you are using this class you *might* have to change your Spring Web MVC configuration slightly. Refer to the class level Javadocs of the UrlFilenameViewController to see examples of the new contract for view name determination.

JPG to PDF Converter For Mac - Unregis

2.8. Updated sample applications

A number of the sample applications have also been updated to showcase the new and improved features of Spring 2.0. So do take the time to investigate them. The aforementioned sample applications can be found in the 'samples' directory of the full Spring distribution ('spring-with-dependencies.[zip|tar.gz]').

Spring 2.5 features revised versions of the PetClinic and PetPortal sample applications, reengineered from the ground up for leveraging Spring 2.5's annotation configuration features. It also uses Java 5 autoboxing, generics, varargs and the enhanced for loop. A Java 5 or 6 SDK is now required to build and run the sample. Check out PetClinic and PetPortal to get an impression of what Spring 2.5 has to offer!

2.9. Improved documentation

The Spring reference documentation has also substantially been updated to reflect all of the above features new in Spring 2.0 and 2.5. While every effort has been made to ensure that there are no errors in this documentation, some errors may nevertheless have crept in. If you do spot any typos or even more serious errors, and you can spare a few cycles during lunch, please do bring the error to the attention of the Spring team by raising an issue.

Difference b/w Spring 2.5 and Spring 3.0

Spring framework 3.0	Spring framework 2.5
Spring framework 3.0 is compatible with Java 5 and higher versions	Spring framework 2.5 is compatible with java 1.4 and higher versions
It introduces Spring expression language which defines bean definitions based XML spring expression and annotation	In this native expression language exists which is less powerful than language of spring 3.0 and it has different parsing rules
spring 3.0 has type converting system and field formatting	does not support type conversions and field formatting
fully supports the JSR303 bean validation	does not offer complete support API
Comprehensive REST support is available	does not
Spring 3.0 automatically validates the @Controller inputs	Does not validate the @Controller inputs

JPG to PDF Converter For Mac - Unregis

Q. Explain DI or IOC pattern.

A: Dependency injection (DI) is a programming design pattern and architectural model, sometimes also referred to as inversion of control or IOC, although technically speaking, dependency injection specifically refers to an implementation of a particular form of IOC. Dependency Injection describes the situation where one object uses a second object to provide a particular capacity. For example, being passed a database connection as an argument to the constructor instead of creating one internally. The term "Dependency injection" is a misnomer, since it is not a dependency that is injected, rather it is a provider of some capability or resource that is injected. There are three common forms of dependency injection: setter-, constructor- and interface-based injection. Dependency injection is a way to achieve loose coupling. Inversion of control (IOC) relates to the way in which an object obtains references to its dependencies. This is often done by a lookup method. The advantage of inversion of control is that it decouples objects from specific lookup mechanisms and implementations of the objects it depends on. As a result, more flexibility is obtained for production applications as well as for testing.

Q. What are the different IOC containers available?

A: Spring is an IOC container. Other IOC containers are HiveMind, Avalon, PicoContainer.

Q. What are the different types of dependency injection. Explain with examples.

A: There are two types of dependency injection: setter injection and constructor injection.
Setter Injection: Normally in all the java beans, we will use setter and getter method to set and get the value of property as follows:

```
public class namebean {
  String     name;
  public void setName(String a) {
     name = a; }
  public String getName() {
     return name; }
  }
```

We will create an instance of the bean 'namebean' (say bean1) and set property as bean1.setName("tom"); Here in setter injection, we will set the property 'name' in spring configuration file as showm below:

```
<bean id="bean1"   class="namebean">
   <property   name="name" >
      <value>tom</value>
   </property>
</bean>
```

The subelement <value> sets the 'name' property by calling the set method as setName("tom"); This process is called setter injection.
To set properties that reference other beans <ref>, subelement of <property> is used as shown below,

```
 <bean id="bean1"   class="bean1impl">
   <property name="game">
      <ref bean="bean2"/>
   </property>
</bean>
<bean id="bean2"   class="bean2impl" />
```


JPG to PDF Converter For Mac - Unregis

Constructor injection: For constructor injection, we use constructor with parameters as shown below,

```
public class namebean {
        String name;
        public namebean(String a) {
           name = a;
        }
    }
```

We will set the property 'name' while creating an instance of the bean 'namebean' as namebean

```
bean1 = new namebean("tom");
```

Here we use the <constructor-arg> element to set the the property by constructor injection as

```
<bean id="bean1"  class="namebean">
   <constructor-arg>
      <value>My Bean Value</value>
   </constructor-arg>
</bean>
```

Q. What is spring? What are the various parts of spring framework? What are the different persistence frameworks which could be used with spring?

A: Spring is an open source framework created to address the complexity of enterprise application development. One of the chief advantages of the Spring framework is its layered architecture, which allows you to be selective about which of its components you use while also providing a cohesive framework for J2EE application development. The Spring modules are built on top of the core container, which defines how beans are created, configured, and managed, as shown in the following figure. Each of the modules (or components) that comprise the Spring framework can stand on its own or be implemented jointly with one or more of the others. The functionality of each component is as follows:

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.

JPG to PDF Converter For Mac - Unregis

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.

Q. What is AOP? How does it relate with IOC? What are different tools to utilize AOP?

A: 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. AOP and IOC are complementary technologies in that both apply a modular approach to complex problems in enterprise application development. In a typical object-oriented development approach you might implement logging functionality by putting logger statements in all your methods and Java classes. In an AOP approach you would instead modularize the logging services and apply them declaratively to the components that required logging. The advantage, of course, is that the Java class doesn't need to know about the existence of the logging service or concern itself with any related code. As a result, application code written using Spring AOP is loosely coupled. The best tool to utilize AOP to its capability is AspectJ. However AspectJ works at he byte code level and you need to use AspectJ compiler to get the aop features built into your compiled code. Nevertheless AOP functionality is fully integrated into the Spring context for transaction management, logging, and various other features. In general any AOP framework control aspects in three possible ways:

Joinpoints: Points in a program's execution. For example, joinpoints could define calls to specific methods in a class

Pointcuts: Program constructs to designate joinpoints and collect specific context at those points

Advices: Code that runs upon meeting certain conditions. For example, an advice could log a message before executing a joinpoint

Q. What are the advantages of spring framework?

A: Spring has layed architecture. Use what you need and leave you don't need now.

1. Spring Enables POJO Programming. There is no behind the scene magic here. POJO programming enables contineous integration and testability.
2. Dependency Injection and Inversion of Control Simplifies JDBC (Read the first question.)
3. Open source and no vendor lock-in.

Q. Can you name a tool which could provide the initial ant files and directory structure for a new spring project.

A: Appfuse or equinox.

JPG to PDF Converter For Mac - Unregis

Q. Explain BeanFactory in spring.

A: Bean factory is an implementation of the factory design pattern and its function is to create and dispense beans. As the bean factory knows about many objects within an application, it is able to create association between collaborating objects as they are instantiated. This removes the burden of configuration from the bean and the client. There are several implementation of BeanFactory. The most useful one is

"org.springframework.beans.factory.xml.XmlBeanFactory" It loads its beans based on the definition contained in an XML file. To create an XmlBeanFactory, pass a InputStream to the constructor. The resource will provide the XML to the factory. BeanFactory factory = new XmlBeanFactory(new FileInputStream("myBean.xml"));

This line tells the bean factory to read the bean definition from the XML file. The bean definition includes the description of beans and their properties. But the bean factory doesn't instantiate the bean yet. To retrieve a bean from a 'BeanFactory', the getBean() method is called. When getBean() method is called, factory will instantiate the bean and begin setting the bean's properties using dependency injection. myBean bean1 = (myBean)factory.getBean("myBean");

Q. Explain the role of ApplicationContext in spring.

A: While Bean Factory is used for simple applications, the Application Context is spring's more advanced container. Like 'BeanFactory' it can be used to load bean definitions, wire beans together and dispense beans upon request. It also provide

1) a means for resolving text messages, including support for internationalization.
2) a generic way to load file resources.
3) events to beans that are registered as listeners.

Because of additional functionality, 'Application Context' is preferred over a BeanFactory. Only when the resource is scarce like mobile devices, 'BeanFactory' is used. The three commonly used implementation of 'Application Context' are

1. 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");

2. 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");

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

Q. How does Spring supports DAO in hibernate?

A: Spring's HibernateDaoSupport class is a convenient super class for Hibernate DAOs. It has handy methods you can call to get a Hibernate Session, or a SessionFactory. The most convenient method is getHibernateTemplate(), which returns a HibernateTemplate. This template wraps Hibernate checked exceptions with runtime exceptions, allowing your DAO interfaces to be Hibernate exception-free.

Example:

```
public class UserDAOHibernate extends HibernateDaoSupport {
public User getUser(Long id) {
    return (User) getHibernateTemplate().get(User.class, id);
    }
    public void saveUser(User user) {
```


JPG to PDF Converter For Mac - Unregis

```
        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);
        }

    }
```

Q. What are the id generator classes in hibernate?

A: increment: It generates identifiers of type long, short or int that are unique only when no other process is inserting data into the same table. It should not the used in the clustered environment.

identity: It supports identity columns in DB2, MySQL, MS SQL Server, Sybase and HypersonicSQL. The returned identifier is of type long, short or int.

sequence: The sequence generator uses a sequence in DB2, PostgreSQL, Oracle, SAP DB, McKoi or a generator in Interbase. The returned identifier is of type long, short or int

hilo: The hilo generator uses a hi/lo algorithm to efficiently generate identifiers of type long, short or int, given a table and column (by default hibernate_unique_key and next_hi respectively) as a source of hi values. The hi/lo algorithm generates identifiers that are unique only for a particular database. Do not use this generator with connections enlisted with JTA or with a user-supplied connection.

seqhilo: The seqhilo generator uses a hi/lo algorithm to efficiently generate identifiers of type long, short or int, given a named database sequence.

uuid: The uuid generator uses a 128-bit UUID algorithm to generate identifiers of type string, unique within a network (the IP address is used). The UUID is encoded as a string of hexadecimal digits of length 32.

guid: It uses a database-generated GUID string on MS SQL Server and MySQL.
native: It picks identity, sequence or hilo depending upon the capabilities of the underlying database.
assigned: lets the application to assign an identifier to the object before save() is called. This is the default strategy if no <generator> element is specified.

select: retrieves a primary key assigned by a database trigger by selecting the row by some unique key and retrieving the primary key value.

foreign: uses the identifier of another associated object. Usually used in conjunction with a <one-to-one> primary key association.

Q. How is a typical spring implementation look like?

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

1. An interface that defines the functions.
2. An Implementation that contains properties, its setter and getter methods, functions etc.,
3. A XML file called Spring configuration file.
4. Client program that uses the function.

JPG to PDF Converter For Mac - Unregis

How do you define hibernate mapping file in spring?

Add the hibernate mapping file entry in mapping resource inside Spring's applicationContext.xml file in the web/WEB-INF directory.

```
<property name="mappingResources">
        <list>
            <value>org/appfuse/model/User.hbm.xml</value>
        </list>
    </property>
```

How do you configure spring in a web application?

It is very easy to configure any J2EE-based web application to use Spring. At the very least, you can simply add Spring's ContextLoaderListener to your web.xml file:

```
<listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-
    class>
    </listener>
```

Can you have xyz.xml file instead of applicationcontext.xml?

ContextLoaderListener is a ServletContextListener that initializes when your webapp starts up. By default, it looks for Spring's configuration file at WEB-INF/applicationContext.xml. You can change this default value by specifying a <context-param> element named "contextConfigLocation." Example:

```
<listener>
   <listener-class>org.springframework.web.context.ContextLoaderListener


<context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/xyz.xml</param-value>
   </context-param>


</listener-class>
      </listener>
```

How do you configure your database driver in spring?

Using datasource "org.springframework.jdbc.datasource.DriverManagerDataSource". Example:

```
<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
   <property name="driverClassName">
      <value>org.hsqldb.jdbcDriver</value>
   </property>
   <property name="url">
      <value>jdbc:hsqldb:db/appfuse</value>
   </property>
   <property name="username"><value>sa</value></property>
   <property name="password"><value></value></property>
</bean>
```


JPG to PDF Converter For Mac - Unregis

Q. How can you configure JNDI instead of datasource in spring applicationcontext.xml?

A: Using "org.springframework.jndi.JndiObjectFactoryBean". Example:

```
<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
   <property name="jndiName">
      <value>java:comp/env/jdbc/appfuse</value>
   </property>
</bean>
```

Q. What are the key benifits of Hibernate?

A: These are the key benifits of Hibernate:

- Transparent persistence based on POJOs without byte code processing
- Powerful object-oriented hibernate query language
- Descriptive O/R Mapping through mapping file.
- Automatic primary key generation
- Hibernate cache : Session Level, Query and Second level cache.
- Performance: Lazy initialization, Outer join fetching, Batch fetching

Q. What is hibernate session and session factory? How do you configure sessionfactory in spring configuration file?

A: Hibernate Session is the main runtime interface between a Java application and Hibernate. SessionFactory allows applications to create hibernate session by reading hibernate configurations file hibernate.cfg.xml.

```
// Initialize the Hibernate environment
      Configuration cfg = new Configuration().configure();
// Create the session factory
      SessionFactory factory = cfg.buildSessionFactory();
// Obtain the new session object
      Session session = factory.openSession();
```

The call to Configuration().configure() loads the hibernate.cfg.xml configuration file and initializes the Hibernate environment. Once the configuration is initialized, you can make any additional modifications you desire programmatically. However, you must make these modifications prior to creating the SessionFactory instance. An instance of SessionFactory is typically created once and used to create all sessions related to a given context.

The main function of the Session is to offer create, read and delete operations for instances of mapped entity classes. Instances may exist in one of three states:

transient: never persistent, not associated with any Session
persistent: associated with a unique Session
detached: previously persistent, not associated with any Session

A Hibernate Session object represents a single unit-of-work for a given data store and is opened by a SessionFactory instance. You must close Sessions when all work for a transaction is completed. The following illustrates a typical Hibernate session:

JPG to PDF Converter For Mac - Unregis

```
Session session = null;
UserInfo user = null;
Transaction tx = null;
try {
  session = factory.openSession();
  tx = session.beginTransaction();
  user = (UserInfo)session.load(UserInfo.class, id);
  tx.commit();
} catch(Exception e) {
  if (tx != null) {
    try {
      tx.rollback();
    } catch (HibernateException e1) {
      throw new DAOException(e1.toString()); }
  } throw new DAOException(e.toString());
} finally {
  if (session != null) {
    try {
      session.close();
    } catch (HibernateException e) { }
  }
}
```

Q. What is the difference between hibernate get and load methods?

A: The load() method is older; get() was added to Hibernate's API due to user request. The difference is trivial:

The following Hibernate code snippet retrieves a User object from the database: User user = (User) session.get(User.class, userID);

The get() method is special because the identifier uniquely identifies a single instance of a class. Hence it's common for applications to use the identifier as a convenient handle to a persistent object. Retrieval by identifier can use the cache when retrieving an object, avoiding a database hit if the object is already cached.

Hibernate also provides a load() method: User user = (User) session.load(User.class, userID);

If load() can't find the object in the cache or database, an exception is thrown. The load() method never returns null. The get() method returns null if the object can't be found. The load() method may return a proxy instead of a real persistent instance. A proxy is a placeholder instance of a runtime-generated subclass (through cglib or Javassist) of a mapped persistent class, it can initialize itself if any method is called that is not the mapped database identifier getter-method. On the other hand, get() never returns a proxy. Choosing between get() and load() is easy: If you're certain the persistent object exists, and nonexistence would be considered exceptional, load() is a good option. If you aren't certain there is a persistent instance with the given identifier, use get() and test the return value to see if it's null. Using load() has a further implication: The application may retrieve a valid reference (a proxy) to a persistent instance without hitting the database to retrieve its persistent state. So load() might not throw an exception when it doesn't find the persistent object in the cache or database; the exception would be thrown later, when the proxy is accessed.

JPG to PDF Converter For Mac - Unregis

Q. What type of transaction management is supported in hibernate?

A: Hibernate communicates with the database via a JDBC Connection; hence it must support both managed and non-managed transactions.

non-managed in web containers:

```
<bean id="transactionManager"
    class="org.springframework.orm.hibernate.HibernateTransactionManager">
      <property name="sessionFactory">
        <ref local="sessionFactory"/>
      </property>
    </bean>
```

managed in application server using JTA:

```
<bean id="transactionManager"
    class="org.springframework.transaction.jta.JtaTransactionManager.">
      <property name="sessionFactory">
        <ref local="sessionFactory"/>
      </property>
    </bean>
```

Q. What is lazy loading and how do you achieve that in hibernate?

A: Lazy setting decides whether to load child objects while loading the Parent Object. You need to specify parent class.Lazy = true in hibernate mapping file. By default the lazy loading of the child objects is true. This make sure that the child objects are not loaded unless they are explicitly invoked in the application by calling getChild() method on parent. In this case hibernate issues a fresh database call to load the child when getChild() is actully called on the Parent object. But in some cases you do need to load the child objects when parent is loaded. Just make the lazy=false and hibernate will load the child when parent is loaded from the database. Examples: Address child of User class can be made lazy if it is not required frequently. But you may need to load the Author object for Book parent whenever you deal with the book for online bookshop.

Hibernate does not support lazy initialization for detached objects. Access to a lazy association outside of the context of an open Hibernate session will result in an exception.

Q. What are the different fetching strategy in Hibernate?

A: Hibernate3 defines the following fetching strategies:

Join fetching - Hibernate retrieves the associated instance or collection in the same SELECT, using an OUTER JOIN.

Select fetching - a second SELECT is used to retrieve the associated entity or collection. Unless you explicitly disable lazy fetching by specifying lazy="false", this second select will only be executed when you actually access the association.

Subselect fetching - a second SELECT is used to retrieve the associated collections for all entities retrieved in a previous query or fetch. Unless you explicitly disable lazy fetching by specifying lazy="false", this second select will only be executed when you actually access the association.Batch fetching - an optimization strategy for select fetching - Hibernate retrieves a batch of entity instances or collections in a single SELECT, by specifying a list of primary keys or foreign keys.

JPG to PDF Converter For Mac - Unregis

For more details read short primer on fetching strategy at http://www.hibernate.org/315.html

Q. What are different types of cache hibernate supports ?

A: Caching is widely used for optimizing database applications. Hibernate uses two different caches for objects: first-level cache and second-level cache. First-level cache is associated with the Session object, while second-level cache is associated with the Session Factory object. By default, Hibernate uses first-level cache on a per-transaction basis. Hibernate uses this cache mainly to reduce the number of SQL queries it needs to generate within a given transaction. For example, if an object is modified several times within the same transaction, Hibernate will generate only one SQL UPDATE statement at the end of the transaction, containing all the modifications. To reduce database traffic, second-level cache keeps loaded objects at the Session Factory level between transactions. These objects are available to the whole application, not just to the user running the query. This way, each time a query returns an object that is already loaded in the cache, one or more database transactions potentially are avoided. In addition, you can use a query-level cache if you need to cache actual query results, rather than just persistent objects. The query cache should always be used in conjunction with the second-level cache. Hibernate supports the following open-source cache implementations out-of-the-box:

- EHCache is a fast, lightweight, and easy-to-use in-process cache. It supports read-only and read/write caching, and memory- and disk-based caching. However, it does not support clustering.
- OSCache is another open-source caching solution. It is part of a larger package, which also provides caching functionalities for JSP pages or arbitrary objects. It is a powerful and flexible package, which, like EHCache, supports read-only and read/write caching, and memory- and disk-based caching. It also provides basic support for clustering via either JavaGroups or JMS.
- SwarmCache is a simple cluster-based caching solution based on JavaGroups. It supports read-only or nonstrict read/write caching (the next section explains this term). This type of cache is appropriate for applications that typically have many more read operations than write operations.
- JBoss TreeCache is a powerful replicated (synchronous or asynchronous) and transactional cache. Use this solution if you really need a true transaction-capable caching architecture.
- Commercial Tangosol Coherence cache.

Q. What are the different caching strategies?

A: The following four caching strategies are available:

- Read-only: This strategy is useful for data that is read frequently but never updated. This is by far the simplest and best-performing cache strategy.
- Read/write: Read/write caches may be appropriate if your data needs to be updated. They carry more overhead than read-only caches. In non-JTA environments, each transaction should be completed when Session.close() or Session.disconnect() is called.
- Nonstrict read/write: This strategy does not guarantee that two transactions won't simultaneously modify the same data. Therefore, it may be most appropriate for data that is read often but only occasionally modified.
- Transactional: This is a fully transactional cache that may be used only in a JTA environment.

JPG to PDF Converter For Mac - Unregis

Q. How do you configure 2nd level cach in hibernate?

A: To activate second-level caching, you need to define the hibernate.cache.provider_class property in the hibernate.cfg.xml file as follows: <hibernate-configuration>

```
<session-factory>
    <property
name="hibernate.cache.provider_class">org.hibernate.cache.EHCacheProvider</property>
  </session-factory>
</hibernate-configuration>
```

By default, the second-level cache is activated and uses the EHCache provider. To use the query cache you must first enable it by setting the property hibernate.cache.use_query_cache to true in hibernate.properties.

Q. What is the difference between sorted and ordered collection in hibernate?

A: A sorted collection is sorted in-memory using java comparator, while order collection is ordered at the database level using order by clause.

Q. What are the types of inheritence models and describe how they work like vertical inheritence and horizontal?

A: There are three types of inheritance mapping in hibernate :
Example: Let us take the simple example of 3 java classes. Class Manager and Worker are inherited from Employee Abstract class.
1. Table per concrete class with unions : In this case there will be 2 tables. Tables: Manager, Worker [all common attributes will be duplicated]
2. Table per class hierarchy: Single Table can be mapped to a class hierarchy. There will be only one table in database called 'Employee' that will represent all the attributes required for all 3 classes. But it needs some discriminating column to differentiate between Manager and worker;
3. Table per subclass: In this case there will be 3 tables represent Employee, Manager and Worker

Additional Questions:

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

JPG to PDF Converter For Mac - Unregis

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 ?

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

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 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.

JPG to PDF Converter For Mac - Unregis

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

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 with Hibernate, JDO and iBATIS

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.

7. 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.

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.

JPG to PDF Converter For Mac - Unregis

- 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 ?

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

12. 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");
  ```

- 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.

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.

JPG to PDF Converter For Mac - Unregis

- 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 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 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>
```


JPG to PDF Converter For Mac - Unregis

```
        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.

JPG to PDF Converter For Mac - Unregis

22. What are ORM's Spring supports ?

Spring supports the following ORM's :

- Hibernate
- iBatis
- JPA (Java 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:

singleton

Scopes a single bean definition to a single object instance per Spring IoC container.

prototype

Scopes a single bean definition to any number of object instances.

request

Scopes a single bean definition to the lifecycle of a single HTTP request; that is each 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

JPG to PDF Converter For Mac - Unregis

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

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

32. 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:

JPG to PDF Converter For Mac - Unregis

•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 abstractions.

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.

36. 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.

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.

JPG to PDF Converter For Mac - Unregis

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 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 between SQLExceptions and Spring's own data-access-strategy-agnostic org.springframework.dao.DataAccessException.

41. What is Spring's JdbcTemplate ?

Spring's JdbcTemplate is central class to interact with a database through JDBC. JdbcTemplate provides many convenience methods for doing things such as converting database data into primitives or objects, executing prepared and callable statements, and providing custom database error handling.

JdbcTemplate template = new JdbcTemplate(myDataSource);

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 ?

Transaction management

EJB:

- Must use a JTA transaction manager.
- Supports transactions that span remote method calls.

Spring:

- Supports multiple transaction environments through its PlatformTransactionManager interface including JTA, Hibernate, JDO, and JDBC.

JPG to PDF Converter For Mac - Unregis

• Does not natively support distributed transactions—it must be used with a JTA transactio manager.

Declarative transaction support

EJB:

•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.

Spring:

•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

EJB:

Supports programmatic bean-managed persistence and declarative container managed persistence.

Spring:

Provides a framework for integrating with several persistence technologies, including JDBC, Hibernate, JDO, and iBATIS.

Declarative security

EJB:

- 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.

Spring:

- No security implementation out-of-the box.
- Acegi, an open source security framework built on top of Spring, provides declarative security through the Spring configuration file or class metadata.

Distributed computing

EJB:

Provides container-managed remote method calls.

Spring:

Provides proxying for remote calls via RMI, JAX-RPC, and web services.

46. Do I need any other SOAP framework to run Spring Web Services?

You don't need any other SOAP framework to use Spring Web services, though it can use some of the features of Axis 1 and 2.

47 . I get NAMESPACE_ERR exceptions when using Spring-WS. What can I do about it?

If you get the following Exception:

JPG to PDF Converter For Mac - Unregis

NAMESPACE_ERR: An attempt is made to create or change an object in a way which is incorrect with regard to namespaces.

Most often, this exception is related to an older version of Xalan being used. Make sure to upgrade to 2.7.0.

48. Does Spring-WS run under Java 1.3?

Spring Web Services requires Java 1.4 or higher.

49. Does Spring-WS work under Java 1.4?

Spring Web Services works under Java 1.4, but it requires some effort to make it work. Java 1.4 is bundled with the older XML parser Crimson, which does not handle namespaces correctly. Additionally, it is bundled with an older version of Xalan, which also has problems. Unfortunately, placing newer versions of these on the class path does not override them.

The only solution that works is to add newer versions of Xerces and Xalan in the lib/endorsed directory of your JDK, as explained in those FAQs (i.e.$JAVA_HOME/lib/endorsed). The following libraries are known to work with Java 1.4.2:

Library	Version
Xerces	2.8.1
Xalan	2.7.0
XML-APIs	1.3.04
SAAJ	1.2

If you want to use WS-Security, note that the XwsSecurityInterceptor requires Java 5, because an underlying library (XWSS) requires it. Instead, you can use the Wss4jSecurityInterceptor.

50. Does Spring-WS work under Java 1.6?

Java 1.6 ships with SAAJ 1.3, JAXB 2.0, and JAXP 1.4 (a custom version of Xerces and Xalan). Overriding these libraries by putting different version on the classpath will result in various classloading issues, or exceptions in org.apache.xml.serializer.ToXMLSAXHandler. The only option for using more recent versions is to put the newer version in the endorsed directory (see above).

51. Why do the Spring-WS unit tests fail under Mac OS X?

For some reason, Apple decided to include a Java 1.4 compatibility jar with their JDK 1.5. This jar includes the XML parsers which were included in Java 1.4. No other JDK distribution does this, so it is unclear what the purpose of this compatibility jar is.

The jar can be found at /System/Library/Frameworks/JavaVM.framework/Versions/1.5.0/Classes/.compatibility/14compatibility.jar. You can safely remove or rename it, and the tests will run again.

52. What is SAAJ?

SAAJ is the SOAP with Attachments API for Java. Like most Java EE libraries, it consists of a set of interfaces (saaj-api.jar), and implementations (saaj-impl.jar). When running in a Application Server, the implementation is typically provided by the application server. Previously, SAAJ has been part of JAXM, but it has been released as a seperate API as part of the , and also as part of J2EE 1.4. SAAJ is generally known as the packagejavax.xml.soap.

Spring-WS uses this standard SAAJ library to create representations of SOAP messages. Alternatively, it can use

JPG to PDF Converter For Mac - Unregis

59. Why does Spring-WS only support contract-first?

Note that Spring-WS only requires you to write the XSD; the WSDL can be generated from that.

60. How do I retrieve the WSDL from a Service?

The &WSDL query parameter does not work.

The &WSDL query parameter is a way to get a WSDL of a class. In SWS, a service is generally not implemented as a single class, but as a collection of endpoints.

There are two ways to expose a WSDL:

•Simply add the WSDL to the root of the WAR, and the file is served normally. This has the disadvantage that the "location" attribute in the WSDL is static, i.e. it does not necessarily reflect the host name of the server. You can transform locations by using a WsdlDefinitionHandlerAdapter.

•Use theMessageDispatcherServlet, which is done is the samples. Every WsdlDefinition listed in the *-servlet.xml will be exposed under the bean name. So if you define a WsdlDefinition namedecho, it will be exposed as echo.wsdl (i.e.http://localhost:8080/echo/echo.wsdl).

61. 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.

62. 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.

63. 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.

64. 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.

65. 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 theSpring configuration file.

66. 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

JPG to PDF Converter For Mac - Unregis

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")
```

67. 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.

68. Explain Bean lifecycle in Spring framework?

•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.

69. 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.

70. 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>
```

71. 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

JPG to PDF Converter For Mac - Unregis

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>
```

72. 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.

73. 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>
```

74. 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.

75. What are the different types of bean injections?

There are two types of bean injections.

- By setter
- By constructor

76. 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>
```

77. What are different types of Autowire types?

There are four different types by which autowiring can be done.

?byName

?byType

?constructor

?autodetect

JPG to PDF Converter For Mac - Unregis

78. 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.

79. 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.

80. 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.

81. 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 joinpoints.

82. 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 theadvice can be applied.

83. 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.

84. 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.

85. 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.

86. 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 intothe target object at the specified joinpoints.

87. What are the different points where weaving can be applied?

- Compile Time
- Classload Time

JPG to PDF Converter For Mac - Unregis

- Runtime

88. 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

89. What are the different types of AutoProxying?

- BeanNameAutoProxyCreator
- DefaultAdvisorAutoProxyCreator
- Metadata autoproxying

90. 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.

91. What is DataAccessException?

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

92. 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>
```

93. 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>
```


JPG to PDF Converter For Mac - Unregis

```
    <property name="password">
     <value>${db.password}</value>
    </property>
  </bean>
```

94. 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.

95. 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>
```


JPG to PDF Converter For Mac - Unregis

96. 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.

97. 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.

98. 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.

99. 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);
```

100. 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.

JPG to PDF Converter For Mac - Unregis