PART-2

Spring-AOP

Spring-MVC

AOP(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. This first section of this article looks into the various terminologies that are commonly used in the ***AOP Environment***. Then it moves into the support that is available in the Spring API for embedding ***Aspects*** into an Application.

2) Introduction to AOP

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 = existingAccount + depositAccount;
        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.

Account.java

```
public class Account{
```

```
    public long deposit(long depositAmount){

        User user = getContext().getUser();
        if (user.getRole().equals("BankAdmin"){
            newAmount = existingAccount + depositAccount;
            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;
    }
}
```

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();
        try{
            User user = getContext().getUser();
            if (user.getRole().equals("BankAdmin"){
                newAmount = existingAccount + depositAccount;
                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;

    }
}
```

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 andTransaction 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
    // exception handling logic

}

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.

Q) What does service layer comprises of?

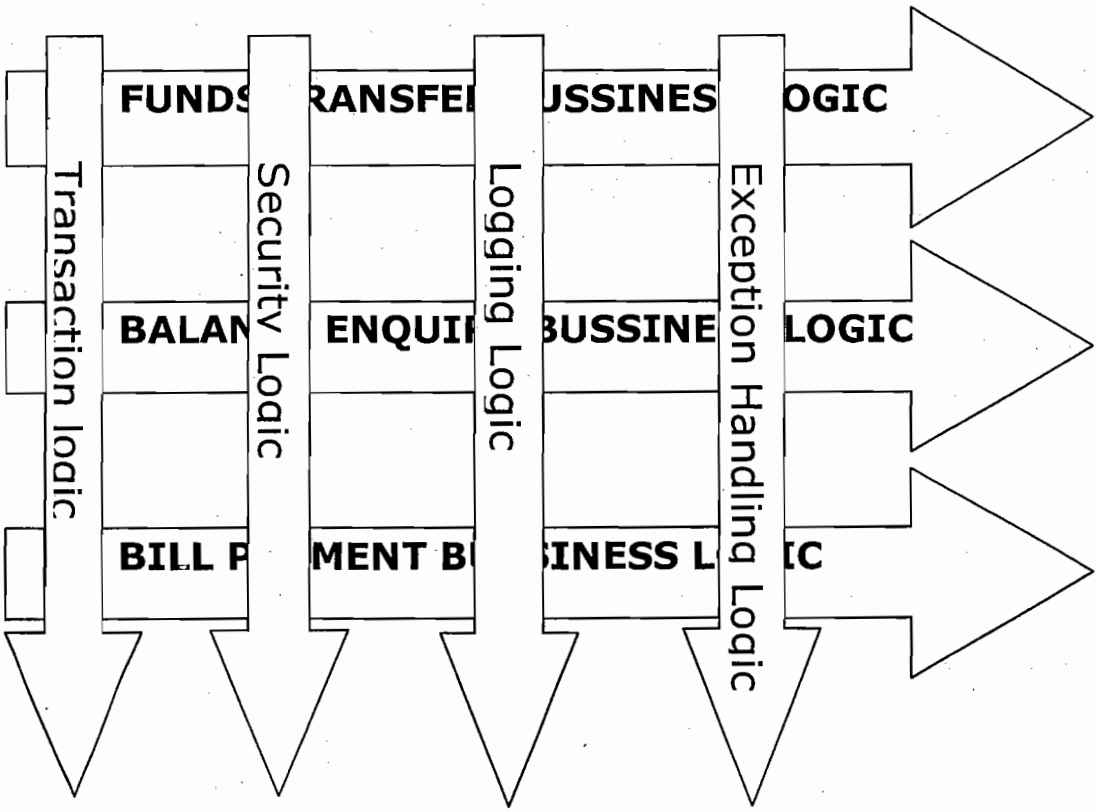
- ⇨ Service layer comprises of two concern
  1) Core concern 2) cross Cutting concern
- ⇨ As far spring application is concern the "concern " mean the piece of code that Perform the well defined task.

Q) What is core concern in service layer?

- ⇨ Programmatically implementation of business rule is business logic. The code that implement rule is nothing but core concern.eg interest Calculation logic .

Q) What is crosscutting concern?

- ⇨ Code that cut the core concern & affect the entire service layer are known as cross cutting concern.
- ⇨ Transaction logic, security logic are the example of the cross cutting code.



Q) What is contribution of Object orientated programming in service layer development?

- ⇨ It is used to built the primary concern ie. Core concern

Q) Can we use Oops for the implementation of cross cutting concern?

- ⇨ Yes! It is static approach of integrating cross cutting concern with core concern.

But it is tightly coupled as far as concern is concern.

Q) What is role of dependency injection in spring application?

- ⇨ DI is meant for decoupling business object (core concern).
- ⇨ Using the DI cross cutting cannot decoupled with core concern.

Q) How to decouple dynamically core concern with crosscutting concerns?

- ⇨ Using AOP (Aspect Orientated programming).

Q) What is AOP?

- ⇨ It is specialized programming methodology using which cross cutting concern is dynamically decoupled from core concern.
- ⇨ AOP is not replacement for OOP.
- ⇨ AOP compliments OOP but does not with compete with AOP.
- ⇨ AOP runs top on the OOP.

Note: - core concern is modularized using OOP. Whereas cross cutting concern are modularized using AOP.

Q.) How the service layer of the spring application is built?

- ⇨ Core concern is modularized using Objects (OOP).
- ⇨ Cross cutting concern are modularized using aspects (AOP).

Business layer in spring = objects + aspects

- ⇨ Core concerns are clearly separated using OOP.
- ⇨ Cross cutting concerns clearly separated and dynamically integrated with core concerns using AOP.

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.

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 likeStart of the Method, End of the Method, the Exception Block, the Finally Block where a particular piece of Aspectcan be made to execute. Such Possible Execution Points in the Application code for embedding Aspects are calledJoin Points. It is not necessary that an Aspect should be applied to all the possible Join Points.

A *single* location in the code where an advice should be executed (such as field access, method invocation , constructor invocation, etc.). Spring's built-in AOP only supports method invocation currently.

NOTE: Spring provides only method call as join point.

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.

}
```

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***.

A pointcut is a set of many joinpoints where an advice should be executed. So if, in Spring, a joinpoint is always a method invocation, then a pointcut is just a set of methods that, when called, should have advices invoked around them.

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***.

In the subsequent sections, we will cover the necessary API in the form of classes and interfaces for supporting the***Aspect Oriented Programming*** in Spring.

3.5) **Target:** - it is business object that comprises of business core concern to which Aspect is applied.

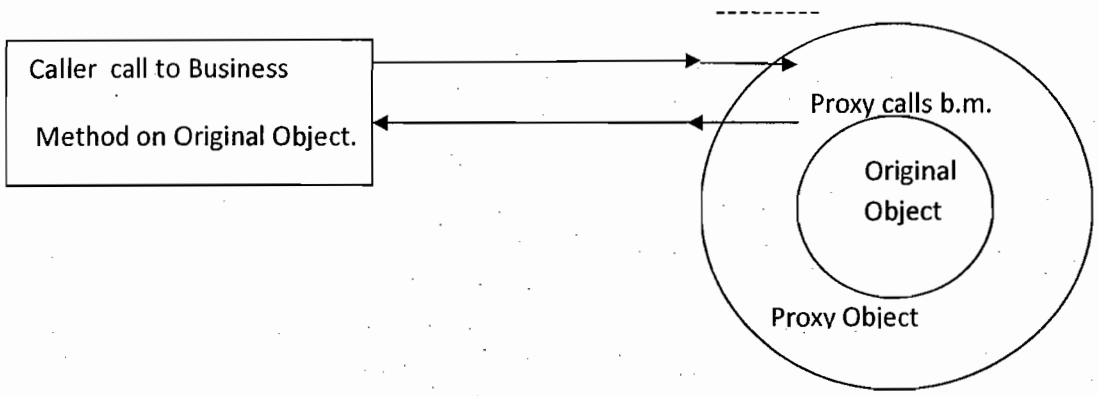
3.6) **Weaving:** - the mechanism of dynamically association of cross cutting concern to Core concern is called Weaving.

3.7) **proxy :** An object produced through weaving is nothing but a proxy.

3.8) **weaver :** code that produces proxies is nothing but weaver.

Q) How does AOP Work?

- ⇨ AOP works on the proxy designed pattern.
  According to proxy design pattern the actual object (business object) is wrapped in to another object knows as proxy & substitute that object in place of actual object (B.O).
- ⇨ Proxy interrupt the call made by caller to original object .it will have chance to whether & when to pass on the call to original object & in mean time any additional code can be executed. There for proxy is best option for cross-cutting code.



There are two type of proxy

1) Static proxy.
2) Dynamic proxy.

- ⇨ Developing and maintaining proxy classes for each business method is static method
- ⇨ Without developing the proxy in advance as when required proxy is developed at run time is known as dynamic proxy.

Spring uses the dynamic proxy approach for AOP.

They are two way to of dynamic proxy generation

1) JDK approach
   a. If the business class is implementing any interface then jdk can create proxy for the business object.
2) cglib approach .
   a. If the business class is not implementing any interface then jdk cannot create proxy for the business object. So here we need the help of cglib (cglib.jar)

Q) What is spring AOP?

Ans. Spring AOP is framework.

- ⇨ Spring AOP is meant for crosscutting concern separation for spring bean registered in IOC container which is business component.
- ⇨ (AOP&IOC goes Hand in hand)
- ⇨ Spring uses dynamic proxy mechanism to provide AOP service spring beans.
- ⇨ Spring framework use jdk,cglib for dynamic proxy creation
- ⇨ Spring 1.x AOP is known as classic spring AOP. It uses the spring interface to implement.
- ⇨ Spring 2.x AOP approach allows us to develop aspect as pojo.

  Meta data are provided by

  1) Annotation & aspectj
  2) Xml element in wirining file.

Q) What are the different kinds of advices in spring AOP?

Ans. 1) Before Advice.

2) After Advice.

3) Around.

4) Throws Advice.

Creating Advices in Spring

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

Steps to configure in configuration file:

- To configure these advices we have to depend on ***ProxyFactoryBean***. This Bean is used to create Proxy objects for the Implementation class along with the Advice implementation.
- Note that the property **'proxyInterfaces'** contains the Interface Name for which the proxy class has to be generated.
- The **'interceptorNames'** property takes a list of Advices to be applied to the dynamically generated proxy class.
- Finally the implementation class is given in the **'target'** property.

1) Before Advice

Before Advice is used to intercept before the method execution starts. In AOP, **BeforeAdvice** is represented in the form of **org.springframework.aop.MethodBeforeAdvice**. 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.

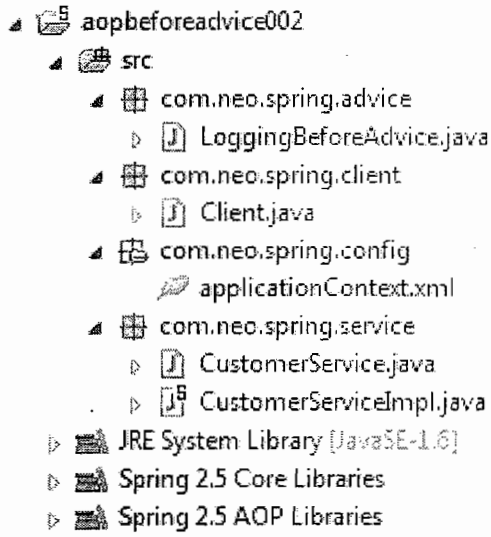
Steps to make use of Before Advice in spring application

Once before advice is applied to a business bean whenever the caller call the business method of the bean first of all before advice code is executed then business method is executed.

Step 1) Develop a business interface & its implementation class i.e. spring bean.
Step 2) Develop a Java class that implements **org.springframework.aop.MethodBeforeAdvice** interface and override **before()** method. In this implement the Crosscutting code.
Step 3) Get the proxy, and make a method call.

To apply BeforeAdvice we should implements ***MethodBeforeAdvice***, thereby telling that **before(Method method, Object[] args, Object target)** 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.

Consider the following example,



CustomerService.java

```
1. package com.neo.spring.service;
2. public interface CustomerService {
3.     String printName();
4.     String printUrl();
5.     void printException();
6. }
```

CustomerServiceImpl.java

```
1. package com.neo.spring.service;
2. public class CustomerServiceImpl implements CustomerService {
3.     private String name;
4.     private String url;
5.
6.         public void setName(String name) {
7.               this.name = name;
8.         }
9.
10.        public void setUrl(String url) {
11.              this.url = url;
12.        }
13.
14.        @Override
15.        public String printName() {
16.              System.out.println("Business Method: printName() =>"+name);
17.              return name;
18.        }
19.
20.        @Override
21.        public String printUrl() {
22.              System.out.println("Business Method: printUrl() =>" + url);
23.              return url;
24.        }
25.
26.        @Override
27.        public void printException() {
28.              throw new IllegalArgumentException("Custom Exception");
29.        }
```

```
30.   }
```

LoggingBeforeAdvice.java

```
1. package com.neo.spring.advice;
2. import java.lang.reflect.Method;
3. import org.springframework.aop.MethodBeforeAdvice;
4.
5. public class LoggingBeforeAdvice implements MethodBeforeAdvice {
6.     @Override
7.     public void before(Method method, Object[] args, Object target)
8.                 throws Throwable {
9.           System.out.println("Before calling :" + method.getName()
10.                + " with arguments : " + args.length + " on :" + target);
11.          }
12.   }
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:p="http://www.springframework.org/schema/p"
5.     xsi:schemaLocation="http://www.springframework.org/schema/beans
6. http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
7.
8.     <bean id="lmba" class="com.neo.spring.advice.LoggingBeforeAdvice" />
9.
10.
11.         <bean id="csImpl"
12.                     class="com.neo.spring.service.CustomerServiceImpl">
13.               <property name="name" value="SomaSekhar"></property>
14.               <property name="url" value="http://neo.com"></property>
15.         </bean>
16.
17.
18.         <!--
19.         <bean id="csProxy"
20.               class="org.springframework.aop.framework.ProxyFactoryBean">
21.               <property name="proxyInterfaces"
22.                         value="com.neo.spring.service.CustomerService" />
23.               <property name="interceptorNames" value="lmba"></property>
24.               <property name="target" ref="csImpl"></property>
25.         </bean>
26.         -->
27.
28.   <bean id="csProxy"
29.               class="org.springframework.aop.framework.ProxyFactoryBean">
30.               <property name="proxyInterfaces">
31.                 <list>
32.                     <value>com.neo.spring.service.CustomerService</value>
33.                 </list>
34.               </property>
35.               <property name="interceptorNames">
36.                     <list>
37.                           <value>lmba</value>
38.                     </list>
```

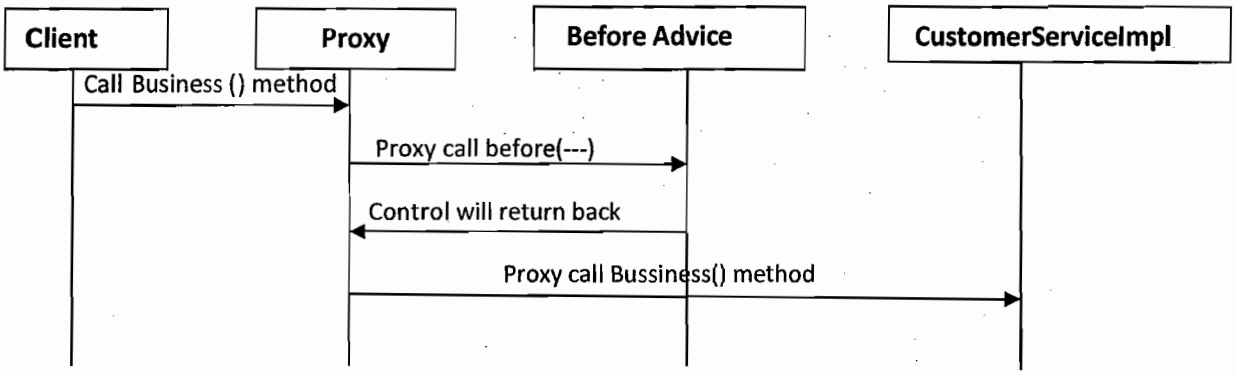
```
39.             </property>
40.             <property name="target" ref="csImpl"></property>
41.         </bean>
42.   </beans>
```

Client.java

```
1. package com.neo.spring.client;
2. import org.springframework.context.ApplicationContext;
3. import org.springframework.context.support.ClassPathXmlApplicationContext;
4. import com.neo.spring.service.CustomerService;
5.
6. public class Client {
7.    private static ApplicationContext context = new
8.         ClassPathXmlApplicationContext(
9.               "com/neo/spring/config/applicationContext.xml");
10.
11.        public static void main(String[] args) {
12.              CustomerService service = (CustomerService)
13.                                              context.getBean("csProxy");
14.              service.printName();
15.              System.out.println();
16.              service.printUrl();
17.              System.out.println();
18.              try {
19.                    service.printException();
20.              } catch (Exception e) {
21.                    System.out.println("Execption raised with message : "
22.                                + e.getMessage());
23.              }
24.        }
25.   }
```

In our application, Client will call business method. but actually call the method on Proxy. Proxy will call the Before Advice method i.e., before() method then control will returns back to Proxy and then Proxy will call the actual business method on CustomerServiceImpl.

Sequence Diagram of Before Advice



Using Before advice, we can perform custom processing before a joinpoint executes. Because a joinpoint in spring is always a method invocation, this essentially allows us to perform preprocessing before the method executes. Before Advice has full access to the target of the method invocation as well as the arguments passed to the method, but it has no control over the execution of the method itself.

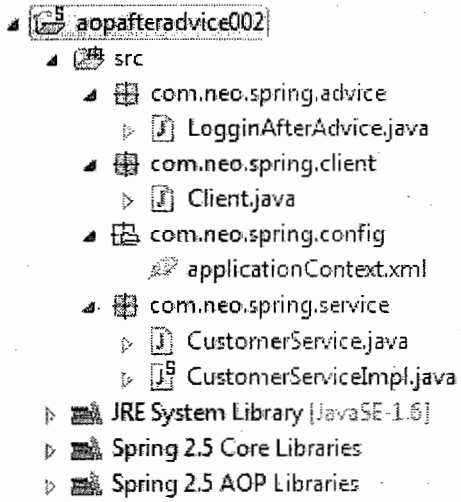
2) After Advice

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***.

Steps to make use of After Advice in spring application

- ⇨ Advice has to implement **org.springframework.aop.AfterReturningAdvice.**
- ⇨ **afterReturning()** method holds the cross cutting code.
- ⇨ Declaration of this method Is
  Public void afterReturning(Object returningvalue, Method M, Object arg[],Object target);
- ⇨ This advice is executed after the successful returning of Business Method.
- ⇨ In this method returning value of business method is available but cant modified it.

Note that, **`afterReturning()`** method that will be called once the method returns normal execution. If some exception happens in the method execution then **`afterReturning()`** method will never be called.



CustomerService.java

```
1. package com.neo.spring.service;
2.
3. public interface CustomerService {
4.     void printName(String name);
5.     String printUrl();
6.     void printException();
7. }
```

CustomerServiceImpl.java

```
1. package com.neo.spring.service;
2.
3. public class CustomerServiceImpl implements CustomerService {
4.          private String url;
5.
6.          public void setUrl(String url) {
7.                this.url = url;
8.          }
9.
```

```
10.         @Override
11.         public void printName(String name) {
12.               System.out.println("Business Method printName() :" + name);
13.         }
14.
15.         @Override
16.         public String printUrl() {
17.               System.out.println("Business Method printUrl() : " + url);
18.               return url;
19.         }
20.
21.         @Override
22.         public void printException() {
23.           throw new IllegalArgumentException("Custom Exception Message");
24.         }
25.   }
```

LogginAfterAdvice

```
1. package com.neo.spring.advice;
2.
3. import java.lang.reflect.Method;
4. import org.springframework.aop.AfterReturningAdvice;
5.
6. public class LogginAfterAdvice implements AfterReturningAdvice {
7.    @Override
8.    public void afterReturning(Object returnValue, Method method,
9.                Object[] args, Object target) throws Throwable {
10.         System.out.println("After calling :" + method.getName() + " on "
11.               + target.getClass() + " With arguments : " + args.length
12.                                 + " giving return value :" + returnValue);
13.               // After calling printName() on CustomerServiceImpl
14.               // with arguments 1, giving return value : null
15.         }
16.   }
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans
3.    xmlns="http://www.springframework.org/schema/beans"
4.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5.    xmlns:p="http://www.springframework.org/schema/p"
6.    xsi:schemaLocation="http://www.springframework.org/schema/beans
7. http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
8.
9.         <bean id="laa" class="com.neo.spring.advice.LogginAfterAdvice" />
10.
11.        <bean id="csImpl"
12.                  class="com.neo.spring.service.CustomerServiceImpl" >
13.              <property name="url" value="http://neo.com" ></property>
14.        </bean>
15.
16.        <bean id="csProxy"
17.             class="org.springframework.aop.framework.ProxyFactoryBean" >
18.              <property name="proxyInterfaces">
19.                <list>
```

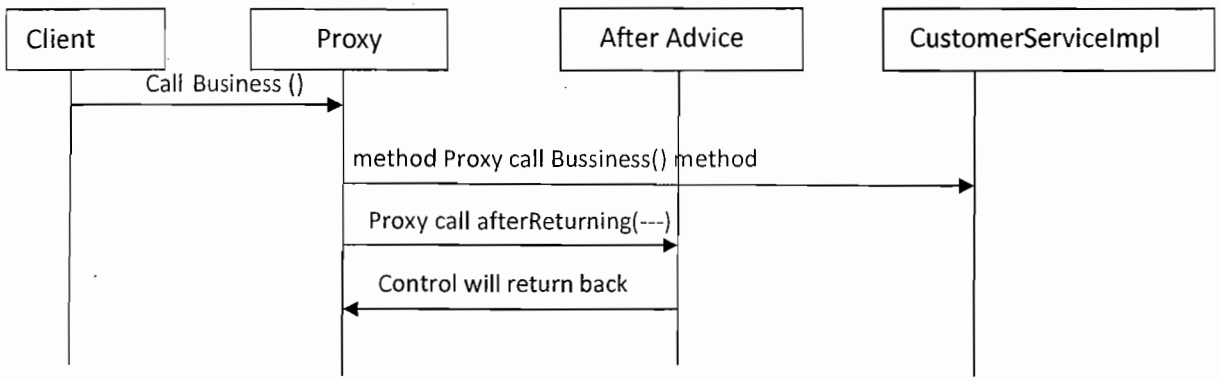
```
20.                     <value>com.neo.spring.service.CustomerService</value>
21.                 </list>
22.             </property>
23.
24.             <property name="interceptorNames">
25.                     <list>
26.                             <value>laa</value>
27.                     </list>
28.             </property>
29.
30.             <property name="target" ref="csImpl"></property>
31.         </bean>
32.
33.   </beans>
```

Client.java

```
1. package com.neo.spring.client;
2. import org.springframework.context.ApplicationContext;
3. import org.springframework.context.support.ClassPathXmlApplicationContext;
4. import com.neo.spring.service.CustomerService;
5.
6. public class Client {
7.     private static ApplicationContext context = new
8.         ClassPathXmlApplicationContext(
9.                 "com/neo/spring/config/applicationContext.xml");
10.
11.         public static void main(String[] args) {
12.             CustomerService service = (CustomerService)
13.                                             context.getBean("csProxy");
14.
15.             service.printName("somasekhar");
16.             System.out.println();
17.             service.printUrl();
18.             System.out.println();
19.             try {
20.                     service.printException();
21.             } catch (Exception e) {
22.                     System.out.println("Error !!! : " + e.getMessage());
23.             }
24.         }
25.   }
```

In our application, Client will calls business method, but actually calls the method on Proxy. Proxy will calls the After Advice method i.e., afterReturning() method then control will returns back to Proxy and then Proxy will call the actual business method on CustomerServiceImpl.

Sequence Diagram of After Advice



Note:- If business method raised Exception it is propagated to client according to the previous advice rules. advice Exception can be raised which is propagated to client preventing value returning to client. afterReturning advice is used to verifies the process data of the business method.

After Returning advice is executed after the method invocation at the joinpoint has finished executing and has returned a value. Then after returning advice has access to the target of the method invocation, the arguments passed to the method, and the return value as well. Because the method has already executed when after returning advice is invoked, it has no control over the method invocation at all.

3) Around Advice

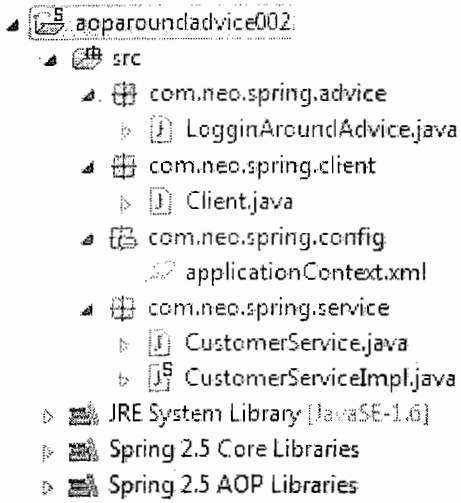
This Advice is very different from the other types of Advice that we have seen before, because of the fact that, this Advice provides finer control whether the target method has to be called or not. Considering the above advices, the return type of the method signature is always `void` meaning that, 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.

Implementing Around Advice:

- ⇨ It is most powerful advice which has the functionality of all three advice with additional capabilities
- ⇨ Around advice 'method is executed before the business method and after the business method. eg. Transaction logic.
- ⇨ Around advice is used in scenario where the before business method execution some crosscutting code to be executed and after the business method successfully or failure execution also crosscutting code is to be executed.
- ⇨ Around class has to implement
  - o **org.aopalliance.intercept.MethodInterceptor** Interface .
- ⇨ This interface has following abstract method has which advice class should implement **public Object invoke (method Invocation mi) throws throwable**

In this method the preprocessing & post processing crosscutting code is implemented.

```
e.g:-  System.out.println("preprocessing crosscutting code ");
       Object O= mi.proceed();
       System.out.println (" postprocessing code");
```



CustomerService.java

```
1. package com.neo.spring.service;
2.
3. public interface CustomerService {
4.     void printName(String name);
5.     String printUrl();
6.     void printException();
7. }
```

CustomerServiceImpl.java

```
1. package com.neo.spring.service;
2.
3. public class CustomerServiceImpl implements CustomerService {
4.          private String url;
5.
6.          public void setUrl(String url) {
7.                this.url = url;
8.          }
9.
10.         @Override
11.         public void printName(String name) {
12.               System.out.println("Business Method printName() :" + name);
13.         }
14.
15.         @Override
16.         public String printUrl() {
17.               System.out.println("Business Method printUrl() : " + url);
18.               return url;
19.         }
20.
21.         @Override
22.         public void printException() {
23.           throw new IllegalArgumentException("Custom Exception Message");
24.         }
25.   }
```

LogginAroundAdvice.java

```
1. package com.neo.spring.advice;
2.
```

```
3. import java.lang.reflect.Method;
4. import org.aopalliance.intercept.MethodInterceptor;
5. import org.aopalliance.intercept.MethodInvocation;
6.
7. public class LogginAroundAdvice implements MethodInterceptor {
8.
9.        @Override
10.       public Object invoke(MethodInvocation invocation) throws Throwable{
11.
12.               Method method = invocation.getMethod();
13.               Object[] args = invocation.getArguments();
14.               Object target = invocation.getThis();
15.               System.out.println("Bofere calling : " + method.getName()
16.                           + " with arguments : " + args.length + " on "
17.                                                   + target.getClass());
18.
19.               Object returnValue = null;
20.
21.               try {
22.                     returnValue = invocation.proceed();
23.               } catch (Exception e) {
24.                     System.out.println("Exception catched in advice : "
25.                                 + e.getMessage());
26.               }
27.
28.               System.out.println("After calling : " + method.getName()
29.                           + " with arguments : " + args.length + " on "
30.                           + target.getClass() + " giving return value : "
31.                                                         + returnValue);
32.               return returnValue;
33.         }
34.   }
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:p="http://www.springframework.org/schema/p"
5.    xsi:schemaLocation="http://www.springframework.org/schema/beans
6. http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
7.
8.          <bean id="laround"
9.                     class="com.neo.spring.advice.LogginAroundAdvice" />
10.
11.         <bean id="csImpl"
12.                    class="com.neo.spring.service.CustomerServiceImpl">
13.               <property name="url" value="http://neo.com"></property>
14.         </bean>
15.
16.         <bean id="csProxy"
17.               class="org.springframework.aop.framework.ProxyFactoryBean">
18.               <property name="proxyInterfaces">
19.                 <list>
20.                     <value>com.neo.spring.service.CustomerService</value>
21.                 </list>
```

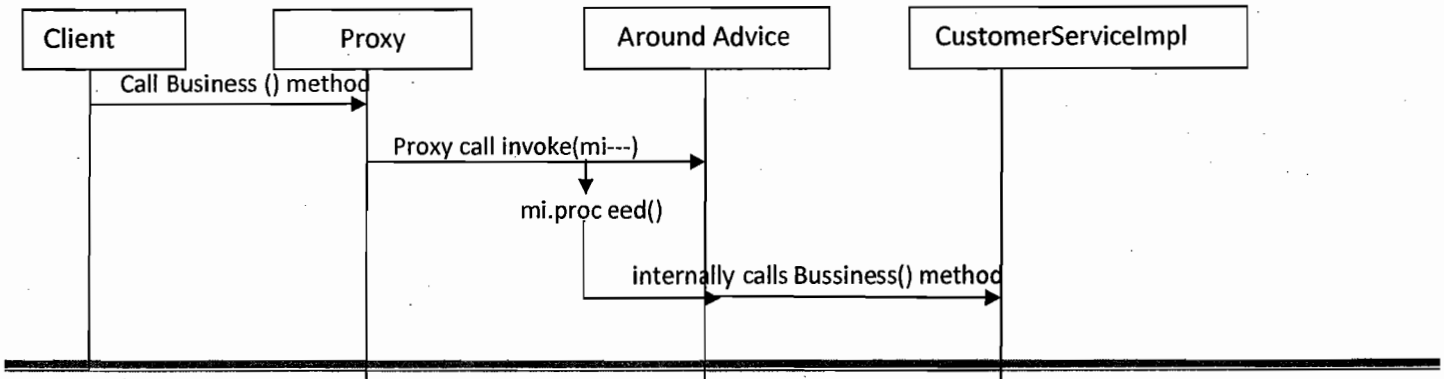
```
22.             </property>
23.
24.             <property name="interceptorNames">
25.                   <list>
26.                         <value>laround</value>
27.                   </list>
28.             </property>
29.
30.             <property name="target" ref="csImpl"></property>
31.       </bean>
32.  </beans>
```

Client.java

```
1. package com.neo.spring.client;
2.
3. import org.springframework.context.ApplicationContext;
4. import org.springframework.context.support.ClassPathXmlApplicationContext;
5. import com.neo.spring.service.CustomerService;
6.
7. public class Client {
8.    private static ApplicationContext context = new
9.            ClassPathXmlApplicationContext(
10.                   "com/neo/spring/config/applicationContext.xml");
11.
12.       public static void main(String[] args) {
13.             CustomerService service = (CustomerService)
14.                                                context.getBean("csProxy");
15.
16.             service.printName("somasekhar");
17.             System.out.println();
18.             service.printUrl();
19.             System.out.println();
20.             service.printException();
21.
22.       }
23.  }
```

Here we will call the business method, but actually it calls the method on proxy, then proxy will call invoke() method, invoke() method will call proceed() method then internally it will call business method. So it has the control over the business method invocation.
It is important to make a call to `MethodInvocation.proceed()` because if we didn't call the proceed() method then it never calls the bussines method.

Sequence Diagram of Around Advice



Around advice is allowed to execute before and after the method invocation, and you can control the point at which the method invocation is allowed to proceed. We can provide our own implementation logic according to our requirement.

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,

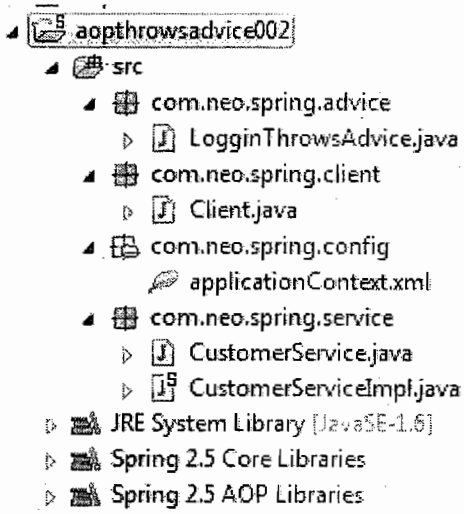
```
public void afterThrowing(Exception ex)

public void afterThrowing(Method method, Object[] args, Object target,
Exception exception)
```

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.

Implementing Throws Advice:

- This advice is executed only when exception is raised business method of advice. Advice has to implement **org.springframework.aop.ThrowsAdvice**.
- It is just marker interface. it don't have any method.
- It is also called null interface.
- Exception handling code is considered as crosscutting code & can be done for entire service layer in Throwsadvice with different kinds of exception parameter.

Note: - Here we have to override the method which take 4 arguments i.e.,
afterThrowing(Method M,Object arg[], object target ,Throwable t)



CustomerService.java

```
1. package com.neo.spring.service;
```

```
2. public interface CustomerService {
3.     void printName(String name);
4.     String printUrl();
5.     void printException();
6. }
```

CustomerServiceImpl.java

```
1. package com.neo.spring.service;
2. public class CustomerServiceImpl implements CustomerService {
3.          private String url;
4.
5.          public void setUrl(String url) {
6.                this.url = url;
7.          }
8.
9.     @Override
10.         public void printName(String name) {
11.               System.out.println("Business Method printName() :" + name);
12.         }
13.
14.         @Override
15.         public String printUrl() {
16.               System.out.println("Business Method printUrl() : " + url);
17.               return url;
18.
19.         }
20.
21.         @Override
22.         public void printException() {
23.           throw new IllegalArgumentException("Custom Exception Message");
24.         }
25.   }
```

LogginThrowsAdvice.java

```
1. package com.neo.spring.advice;
2.
3. import java.lang.reflect.Method;
4. import org.springframework.aop.ThrowsAdvice;
5.
6. public class LogginThrowsAdvice implements ThrowsAdvice {
7.     // public void afterThrowing(Exception ex)
8.
9.     public void afterThrowing(Method method, Object[] args, Object target,
10.                                                    Exception exception) {
11.          System.out.println("After throwing Exception by method : "
12.             + method.getName() + " with argumetns : " + args.length  + "
13.             on " + target.getClass() + " with exception message : "+
14.                                                  exception.getMessage());
15.          }
16.   }
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
4. xmlns:p="http://www.springframework.org/schema/p"
5.     xsi:schemaLocation="http://www.springframework.org/schema/beans
6. http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
7.
8.     <bean id="lta" class="com.neo.spring.advice.LogginThrowsAdvice"></bean>
9.
10.         <bean id="csImpl"
11.                     class="com.neo.spring.service.CustomerServiceImpl">
12.               <property name="url" value="http://neo.com"></property>
13.         </bean>
14.
15.         <bean id="csProxy"
16.               class="org.springframework.aop.framework.ProxyFactoryBean">
17.               <property name="proxyInterfaces">
18.                 <list>
19.                     <value>com.neo.spring.service.CustomerService</value>
20.                 </list>
21.               </property>
22.               <property name="interceptorNames">
23.                     <list>
24.                           <value>lta</value>
25.                     </list>
26.               </property>
27.               <property name="target" ref="csImpl"></property>
28.         </bean>
29.   </beans>
```

Client.java

```
1. package com.neo.spring.client;
2. import org.springframework.context.ApplicationContext;
3. import org.springframework.context.support.ClassPathXmlApplicationContext;
4. import com.neo.spring.service.CustomerService;
5.
6. public class Client {
7.    private static ApplicationContext context = new
8.        ClassPathXmlApplicationContext(
9.               "com/neo/spring/config/applicationContext.xml");
10.
11.        public static void main(String[] args) {
12.              CustomerService service = (CustomerService)
13.                                        context.getBean("csProxy");
14.
15.              service.printName("somasekhar");
16.              System.out.println();
17.              service.printUrl();
18.              System.out.println();
19.              service.printException();
20.
21.        }
22.   }
```

Throws advice is executed after a method invocation returns but only if that invocation threw an exception. It is possible for throws advice to catch only specific exceptions, and if we choose to do so, we can access the method that threw the exception, the arguments passed into the invocation, and the target of the invocation.

Q) Can we alter business method argument in before() method of advice?

⇨ Yes. The modified value only will be supplied to business method of target object as argument.

Q) How to deal with proxy beans?

⇨ Per bean configure the **ProxyfactoryBean** with different ID on bean configuration file. While calling the **getBean()** specify that id to get proxy for corresponding bean.

⇨ We have to follow naming convention to give an ID to the proxy because we configure multiple proxy objects. So to identify particular proxy we have to follow the naming convention.

Q) What are the additional things in around advice can do which cannot be done by Another advice?

⇨ before advice can't prevent to business method() call from being invoked under normal condition. Around advice can do it by returning from **invoke()** with calling **proceed()**.

⇨ After returning advice cannot changes the return value. around advice can.

⇨ Throws advice even handles the exception it cannot prevent the Exception benign propagated to client .around advice can propagated some value to client
Instate of exception even though exception is raised.

Q) How to choose an advice type?

⇨ Least powerful advice that suit your requirement.

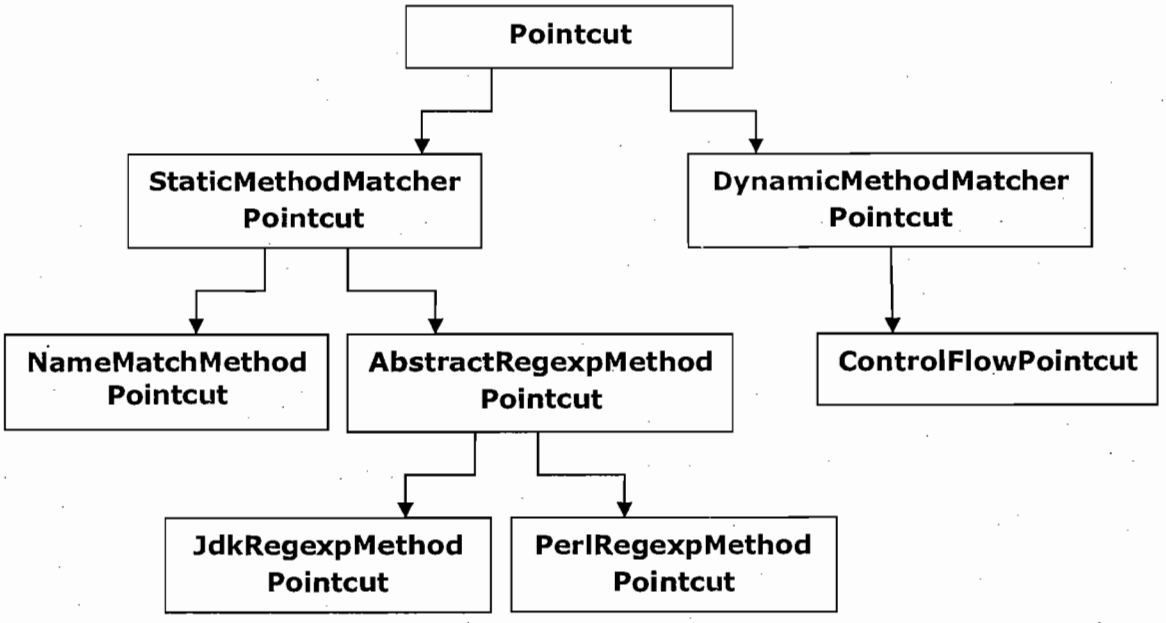
1) Before advice ---------- security logic
2) After Retuning ------ logging verification process data
3) Throws ------- centralizing the exception
4) Around --------- transaction management, (performance compromise)

Pointcut in AOP

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***.

- ⇨ Static point cut verifies whether that join point has to be advised or not, only once the result is catched & reused.
- ⇨ Dynamic point cut verifies every time as it has to decide the join point based on the argument passed to method call.



Point Cut is an interface, represented by `org.springframework.aop.Pointcut`.

```
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) ){
            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)` won't be called. The implementation of the 2 argument `matches()` method essentially says that we want only the getter methods to be advised.

StaticMethodMatcherPointcut has two types of Pointcuts:

- NameMatchMethodPointcut
- RegularExpressionPointcut

In last Spring AOP advice example, all the methods in a class will be intercept automatically. But for most cases, we may need a way to intercept only one or two methods; this is what 'Pointcut' is. It's allows us to intercept a method by it's method name. In addition, a 'Pointcut' must be associated with an '**Advisor**'.

In Spring AOP, comes with three very common technical terms - **Advices, Pointcut, Advisor**, put it in unofficial way...

- Advice – Indicate the action to take either before or after the method execution.
- Pointcut – Indicate which method should be intercept, by method name or regular expression name.
- Advisor – Group 'Advice' and 'Pointcut' into a single unit, and pass it to a proxy factory object.

NameMatchPointCut

We can intercept a method via 'pointcut' and 'advisor'. Create a **NameMatchMethodPointcut** pointcut bean, and put the method name we want to intercept in the '**mappedName**' property value.

Create a **DefaultPointcutAdvisor** advisor bean, and associate both advice and pointcut.

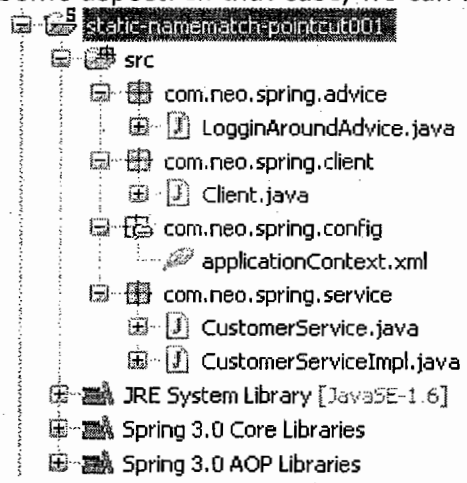
- ⇨ **org.springframework.aop.support.NameMatchMethodPointcut** has to configure in bean configuration file. It has **java.util.List** type variable "**mappedNames**". Through setter injection we need to specify the method names of the target object which we want them to advice.
- ⇨ This point cut is bean uses their names to decide the selection of join point.
- ⇨ This point cut should inform to proxy.
- ⇨ We cannot inform only point cut to **ProxyFactoryBean**. We need to use built in advisor for this purpose.
- ⇨ This advisor represent the association between advice & point cut (aspect)
- ⇨ While configuration the **ProxyFactoryBean** we need to specify the advisor instead of advice.

We can use **NameMatchMethodPointcutAdvisor** to combine both pointcut and advisor into a single bean.

Here the name of the methods that are too be given advices can be directly mentioned in the Configuration File. `'*'` represents that all the methods in the class should be given Advice.

The Expression **`print*`** tells that all method names starting with the method name `print` will be given ***Advices***. If we want all the methods in our Class to be adviced, then the `'value'` tag should be given `'*'` meaning all the methods in the Class.

Suppose we wish that only the methods **`printName()`** and **`printUrl()`** should be given ***Advice*** by some aspect. In that case, we can have the following example,



CustomerService.java

```
1. package com.neo.spring.service;
2.
3. public interface CustomerService {
4. void printName(String name);
5. String printUrl();
6. void printException();
7. }
```

CustomerServiceImpl.java

```
1. package com.neo.spring.service;
2.
3. public class CustomerServiceImpl implements CustomerService {
4.    private String url;
5.
```

```
6.    public void setUrl(String url) {
7.      this.url = url;
8.    }
9.
10.   @Override
11.   public void printName(String name) {
12.      System.out.println("Business Method printName() :" + name);
13.   }
14.
15.   @Override
16.   public String printUrl() {
17.      System.out.println("Business Method printUrl() : " + url);
18.      return url;
19.
20.   }
21.
22.   @Override
23.   public void printException() {
24.      throw new IllegalArgumentException("Custom Exception Message");
25.   }
26.   }
```

LogginAroundAdvice.java

```
1. package com.neo.spring.advice;
2.
3. import java.lang.reflect.Method;
4. import org.aopalliance.intercept.MethodInterceptor;
5. import org.aopalliance.intercept.MethodInvocation;
6.
7. public class LogginAroundAdvice implements MethodInterceptor {
8.
9.    @Override
10.   public Object invoke(MethodInvocation invocation) throws Throwable {
11.
12.             Method method = invocation.getMethod();
13.             Object[] args = invocation.getArguments();
14.             Object target = invocation.getThis();
15.             System.out.println("Before calling : " + method.getName()
16.                       + " with arguments : " + args.length + " on "
17.                       + target.getClass());
18.             Object returnValue = null;
19.
20.             try {
21.                   returnValue = invocation.proceed();
22.             } catch (Exception e) {
23.                   System.out.println("Exception catched in advice : "
24.                             + e.getMessage());
25.             }
26.
27.             System.out.println("After calling : " + method.getName()
28.                       + " with arguments : " + args.length + " on "
29.                       + target.getClass() + " giving return value : "
30.                                                   + returnValue);
31.             return returnValue;
32.       }
```

```
33.   }
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:p="http://www.springframework.org/schema/p"
5.    xsi:schemaLocation="http://www.springframework.org/schema/beans
6. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7.
8.    <bean id="csImpl" class="com.neo.spring.service.CustomerServiceImpl">
9.          <property name="url" value="http://neo.com"></property>
10.   </bean>
11.
12.   <bean id="laround" class="com.neo.spring.advice.LogginAroundAdvice" />
13.
14.   <!-
15.   <bean id="nameUrlPointcut"
16.          class="org.springframework.aop.support.NameMatchMethodPointcut">
17.         <property name="mappedNames">
18.               <list>
19.                     <value>printName</value>
20.                     <value>printUrl</value>
21.               </list>
22.         </property>
23.
24.   </bean>
25.
26.   <bean id="laroundNameUrlAdvisor"
27.          class="org.springframework.aop.support.DefaultPointcutAdvisor">
28.         <property name="advice" ref="laround"></property>
29.         <property name="pointcut" ref="nameUrlPointcut"></property>
30.   </bean>
31.   -->
32.
33.   <bean id="laroundNameUrlAdvisor"
34.   class="org.springframework.aop.support.NameMatchMethodPointcutAdvisor">
35.         <property name="advice" ref="laround" ></property>
36.         <property name="mappedNames">
37.               <list>
38.                     <value>printName</value>
39.                     <value>printUrl</value>
40.               </list>
41.         </property>
42.   </bean>
43.
44.   <bean id="csProxy"
45.               class="org.springframework.aop.framework.ProxyFactoryBean">
46.         <property name="proxyInterfaces">
47.               <list>
48.                     <value>com.neo.spring.service.CustomerService</value>
49.               </list>
50.         </property>
51.
52.         <property name="target" ref="csImpl"></property>
```

```
53.         <property name="interceptorNames">
54.               <list>
55.                     <value>laroundNameUrlAdvisor</value>
56.               </list>
57.         </property>
58.   </bean>
59.
60.   </beans>
```

Client.java

```
1. package com.neo.spring.client;
2.
3. import org.springframework.context.ApplicationContext;
4. import org.springframework.context.support.ClassPathXmlApplicationContext;
5. import com.neo.spring.service.CustomerService;
6.
7. public class Client {
8.    private static ApplicationContext context = new
9.          ClassPathXmlApplicationContext(
10.                   "com/neo/spring/config/applicationContext.xml");
11.
12.         public static void main(String[] args) {
13.               CustomerService service = (CustomerService)
14.                                          context.getBean("csProxy");
15.
16.               service.printName("somasekhar");
17.               System.out.println();
18.               service.printUrl();
19.               System.out.println();
20.               service.printException();
21.         }
22.   }
```

Regular Expression Pointcut

Beside the matching method by name, We can also match the method's name by using regular expression pointcut.

This pointcut is used to verify join point based on pattern of method name instead of name.

This kind is used if we want to match the name of the methods in the Class based on ***Regular Expression***.

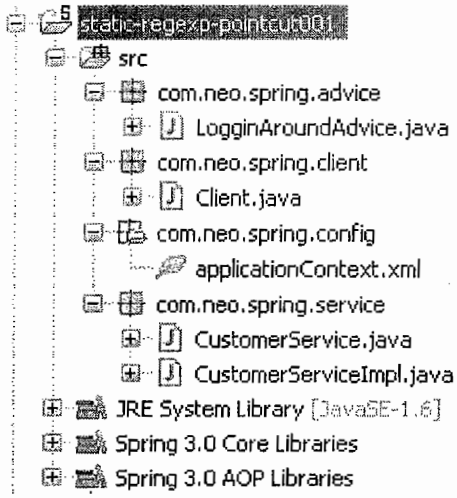
We have two types of regular Expressions:

- ***PerlRegularExpression***
  represented by `org.springframework.aop.support.Perl5RegexpMethodPointcut`
- ***JdkRegularExpression***
  represented by `org.springframework.aop.support.JdkRegexpMethodPointcut`

Here we are using ***JdkRegularExpression***.

The Expression `*.Exception*.` tells that method names ending with the method name `Exception` will be given ***Advices***.

Suppose we wish that only the methods `printUrl()` and `printException()` should be given ***Advice*** by some aspect. In that case, we can have the following example,



CustomerService.java

```
1. package com.neo.spring.service;
2.
3. public interface CustomerService {
4.     void printName(String name);
5.     String printUrl();
6.     void printException();
7. }
```

CustomerServiceImpl.java

```
1. package com.neo.spring.service;
2. public class CustomerServiceImpl implements CustomerService {
3.          private String url;
4.
5.          public void setUrl(String url) {
6.                this.url = url;
7.          }
8.
9.          @Override
10.         public void printName(String name) {
11.               System.out.println("Business Method printName() :" + name);
12.         }
13.
14.         @Override
15.         public String printUrl() {
16.               System.out.println("Business Method printUrl() : " + url);
17.               return url;
18.         }
19.
20.         @Override
21.         public void printException() {
22.           throw new IllegalArgumentException("Custom Exception Message");
23.         }
24.   }
```

LogginAroundAdvice.java

```
1. package com.neo.spring.advice;
2.
3. import java.lang.reflect.Method;
4. import org.aopalliance.intercept.MethodInterceptor;
5. import org.aopalliance.intercept.MethodInvocation;
6.
7. public class LogginAroundAdvice implements MethodInterceptor {
8.
9.     @Override
10.    public Object invoke(MethodInvocation invocation) throws Throwable {
11.
12.                Method method = invocation.getMethod();
13.                Object[] args = invocation.getArguments();
14.                Object target = invocation.getThis();
15.                System.out.println("Before calling : " + method.getName()
16.                            + " with arguments : " + args.length + " on "
17.                            + target.getClass());
18.
19.                Object returnValue = null;
20.
21.                try {
22.                      returnValue = invocation.proceed();
23.                } catch (Exception e) {
24.                      System.out.println("Exception catched in advice : "
25.                                  + e.getMessage());
26.                }
27.
28.                System.out.println("After calling : " + method.getName()
29.                            + " with arguments : " + args.length + " on "
30.                            + target.getClass() + " giving return value : "
31.                                                      + returnValue);
32.                return returnValue;
33.          }
34.    }
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:p="http://www.springframework.org/schema/p"
5.     xsi:schemaLocation="http://www.springframework.org/schema/beans
6. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7.     <bean id="csImpl" class="com.neo.spring.service.CustomerServiceImpl">
8.           <property name="url" value="http://neo.com"></property>
9.     </bean>
10.
11.    <bean id="laround" class="com.neo.spring.advice.LogginAroundAdvice" />
12.
13.    <!--
14.    <bean id="exceptionUrlPc"
15.          class="org.springframework.aop.support.JdkRegexpMethodPointcut">
16.          <property name="patterns">
17.                <list>
18.                      <value>.*Exception.*</value>
```

```
19.                     <value>.*Url.*</value>
20.               </list>
21.         </property>
22.   </bean>
23.
24.   <bean id="laroundExceptionUrlAdvisor"
25.         class="org.springframework.aop.support.DefaultPointcutAdvisor">
26.         <property name="advice" ref="laround"></property>
27.         <property name="pointcut" ref="exceptionUrlPc"></property>
28.   </bean>
29.   -->
30.
31.   <bean id="laroundExceptionUrlAdvisor"
32.   class="org.springframework.aop.support.RegexpMethodPointcutAdvisor" >
33.         <property name="advice" ref="laround" ></property>
34.         <property name="patterns">
35.               <list>
36.                     <value>.*Exception.*</value>
37.                     <value>.*Url.*</value>
38.               </list>
39.         </property>
40.   </bean>
41.
42.   <bean id="csProxy"
43.         class="org.springframework.aop.framework.ProxyFactoryBean" >
44.         <property name="proxyInterfaces">
45.               <list>
46.                     <value>com.neo.spring.service.CustomerService</value>
47.               </list>
48.         </property>
49.
50.         <property name="target" ref="csImpl"></property>
51.         <property name="interceptorNames">
52.               <list>
53.                     <value>laroundExceptionUrlAdvisor</value>
54.               </list>
55.         </property>
56.   </bean>
57.
58.   </beans>
```

Client.java

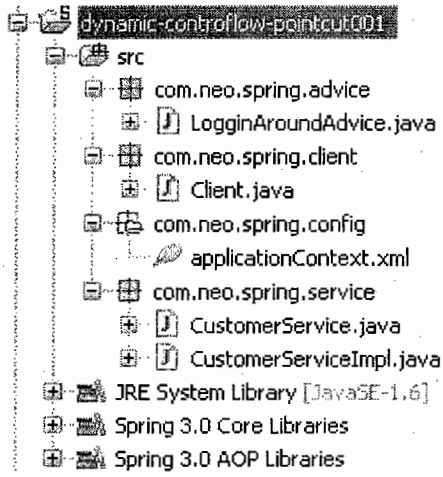
```
1. package com.neo.spring.client;
2.
3. import org.springframework.context.ApplicationContext;
4. import org.springframework.context.support.ClassPathXmlApplicationContext;
5. import com.neo.spring.service.CustomerService;
6.
7. public class Client {
8.    private static ApplicationContext context = new
9.              ClassPathXmlApplicationContext(
10.                   "com/neo/spring/config/applicationContext.xml");
11.
12.         public static void main(String[] args) {
13.               CustomerService service = (CustomerService)
```

```
14.                                         context.getBean("csProxy");
15.
16.             service.printName("somasekhar");
17.             System.out.println();
18.             service.printUrl();
19.             System.out.println();
20.             service.printException();
21.         }
22.   }
```

It will only match the method which has 'Url' within the method name. It's also quite useful for the DAO transaction management, where we can declare ".*DAO.*" to include all our DAO classes.

Dynamic ControFlow Pointcut

This point cut is used to verify from which context business method call is made. If method call is made in specified flow it will be advice otherwise it won't.



CustomerService.java

```
1. package com.neo.spring.service;
2.
3. public interface CustomerService {
4.     void printName(String name);
5.     String printUrl();
6.     void printException();
7. }
```

CustomerServiceImpl.java

```
1. package com.neo.spring.service;
2.
3. public class CustomerServiceImpl implements CustomerService {
4.          private String url;
5.
6.          public void setUrl(String url) {
7.                this.url = url;
8.          }
9.
10.         @Override
```

```
11.         public void printName(String name) {
12.               System.out.println("Business Method printName() :" + name);
13.         }
14.
15.         @Override
16.         public String printUrl() {
17.               System.out.println("Business Method printUrl() : " + url);
18.               return url;
19.
20.         }
21.
22.         @Override
23.         public void printException() {
24.           throw new IllegalArgumentException("Custom Exception Message");
25.         }
26.   }
```

LogginAroundAdvice.java

```
1. package com.neo.spring.advice;
2.
3. import java.lang.reflect.Method;
4. import org.aopalliance.intercept.MethodInterceptor;
5. import org.aopalliance.intercept.MethodInvocation;
6.
7. public class LogginAroundAdvice implements MethodInterceptor {
8.
9.    @Override
10.   public Object invoke(MethodInvocation invocation) throws Throwable {
11.
12.               Method method = invocation.getMethod();
13.               Object[] args = invocation.getArguments();
14.               Object target = invocation.getThis();
15.               System.out.println("Before calling : " + method.getName()
16.                         + " with arguments : " + args.length + " on "
17.                         + target.getClass());
18.               Object returnValue = null;
19.
20.               try {
21.                     returnValue = invocation.proceed();
22.               } catch (Exception e) {
23.                     System.out.println("Exception catched in advice : "
24.                               + e.getMessage());
25.               }
26.
27.               System.out.println("After calling : " + method.getName()
28.                         + " with arguments : " + args.length + " on "
29.                         + target.getClass() + " giving return value : "
30.                                                   + returnValue);
31.               return returnValue;
32.         }
33.   }
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
```

```
3.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:p="http://www.springframework.org/schema/p"
5.    xsi:schemaLocation="http://www.springframework.org/schema/beans
6. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7.
8.    <bean id="csImpl" class="com.neo.spring.service.CustomerServiceImpl">
9.          <property name="url" value="http://neo.com"></property>
10.   </bean>
11.
12.   <bean id="laround" class="com.neo.spring.advice.LogginAroundAdvice" />
13.
14.   <bean id="clientMainPc"
15.            class="org.springframework.aop.support.ControlFlowPointcut">
16.         <constructor-arg type="java.lang.Class" index="0"
17.                     value="com.neo.spring.client.Client" />
18.         <constructor-arg type="java.lang.String" index="1"
19.                     value="main" />
20.   </bean>
21.
22.   <bean id="laroundClientMainAdvisor"
23.          class="org.springframework.aop.support.DefaultPointcutAdvisor" >
24.         <property name="advice" ref="laround" ></property>
25.         <property name="pointcut" ref="clientMainPc" ></property>
26.   </bean>
27.
28.   <bean id="csProxy"
29.              class="org.springframework.aop.framework.ProxyFactoryBean" >
30.         <property name="proxyInterfaces">
31.               <list>
32.                     <value>com.neo.spring.service.CustomerService</value>
33.               </list>
34.         </property>
35.
36.         <property name="target" ref="csImpl" ></property>
37.         <property name="interceptorNames">
38.               <list>
39.                     <value>laroundClientMainAdvisor</value>
40.               </list>
41.         </property>
42.   </bean>
43.
44.   </beans>
```

Client.java

```
1. package com.neo.spring.client;
2.
3. import org.springframework.context.ApplicationContext;
4. import org.springframework.context.support.ClassPathXmlApplicationContext;
5. import com.neo.spring.service.CustomerService;
6.
7. public class Client {
8.    private static ApplicationContext context = new
9.      ClassPathXmlApplicationContext(
10.                    "com/neo/spring/config/applicationContext.xml");
11.
```

```
12.         public static void main(String[] args) {
13.               CustomerService service = (CustomerService)
14.                                         context.getBean("csProxy");
15.
16.               service.printName("somasekhar");
17.               System.out.println();
18.               service.printUrl();
19.               System.out.println();
20.               service.printException();
21.         }
22.   }
```

Schema-Based AOP support

Spring 2.0 also offers support for defining aspects using the new "aop" namespace tags. To use the aop namespace tags we need to import the spring-aop schema.
Within our Spring configurations, all aspect and advisor elements must be placed within an `<aop:config>` element (we can have more than one `<aop:config>` element in an application context configuration). An `<aop:config>` element can contain pointcut, advisor, and aspect elements (note these must be declared in that order).

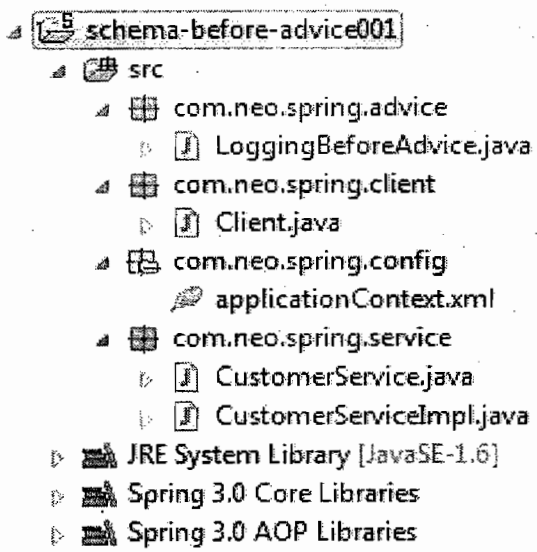
Declaring an aspect
Using the schema support, an aspect is simply a regular Java object defined as a bean in your Spring application context. The state and behavior is captured in the fields and methods of the object, and the pointcut and advice information is captured in the XML.
An aspect is declared using the <aop:aspect> element.

Declaring a pointcut
A named pointcut can be declared inside an <aop:config> element, enabling the pointcut definition to be shared across several aspects and advisors.

BeforeAdvice

Before advice runs before a matched method execution. It is declared inside an `<aop:aspect>` using the <aop:before> element.

The method attribute identifies a method that provides the body of the advice. This method must be defined for the bean referenced by the aspect element containing the advice. Before a data access operation is executed (a method execution join point matched by the pointcut expression), the method on the aspect bean will be invoked.



CustomerService.java

```
1. package com.neo.spring.service;
2. public interface CustomerService {
3.          String printName();
4.          String printUrl();
5.          void printException();
6. }
```

CustomerServiceImpl.java

```
1. package com.neo.spring.service;
2.
3. public class CustomerServiceImpl implements CustomerService {
4.          private String name;
5.          private String url;
6.
7.          public void setName(String name) {
8.                this.name = name;
9.          }
10.
11.    public void setUrl(String url) {
12.          this.url = url;
13.    }
14.
15.    @Override
16.    public String printName() {
17.          System.out.println("Business Method: printName() =>"+name);
18.          return name;
19.    }
20.
21.    @Override
22.    public String printUrl() {
23.          System.out.println("Business Method: printUrl() =>" + url);
24.          return url;
25.    }
26.
27.    @Override
28.    public void printException() {
29.         throw new IllegalArgumentException("Custom Exception");
30.    }
31.    }
```

LoggingBeforeAdvice.java

```
1. package com.neo.spring.advice;
2. public class LoggingBeforeAdvice {
3.        public void myBefore() {
4.          System.out.println("LoggingBeforeAdvice.before");
5.     }
6. }
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2.
3. <beans xmlns="http://www.springframework.org/schema/beans"
4.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xmlns:aop="http://www.springframework.org/schema/aop"
6.     xsi:schemaLocation="http://www.springframework.org/schema/beans
7.          http://www.springframework.org/schema/beans/spring-beans.xsd
8.          http://www.springframework.org/schema/aop
9.          http://www.springframework.org/schema/aop/spring-aop.xsd">
10.
11.    <bean id="csImpl"
12.               class="com.neo.spring.service.CustomerServiceImpl" />
```

```
13.   <bean id="lba"
14.               class="com.neo.spring.advice.LoggingBeforeAdvice" />
15.
16.   <aop:config>
17.   <!--
18.    <aop:pointcut id="csPc"
19.   expression="within(com.neo.spring.service.CustomerServiceImpl)"/>
20.
21.     <aop:aspect ref="lba">
22.         <aop:before method="myBefore" pointcut-ref="csPc" />
23.     </aop:aspect>
24.   -->
25.
26.    <aop:aspect ref="lba">
27.     <aop:before method="myBefore"
28.     pointcut="within(com.neo.spring.service.CustomerServiceImpl)"/>
29.    </aop:aspect>
30.
31.   </aop:config>
32.
33.   </beans>
```

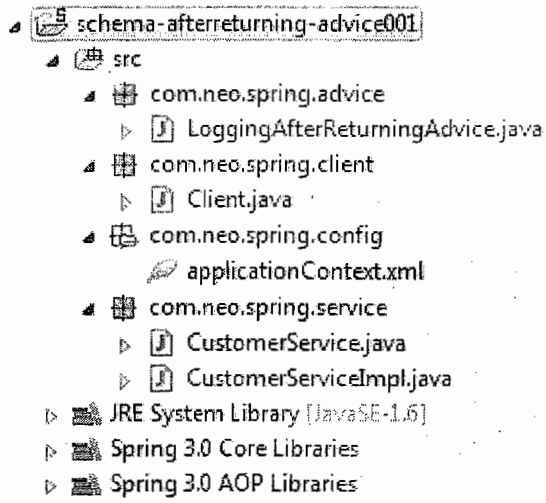
Client.java

```
1. package com.neo.spring.client;
2.
3. import org.springframework.context.ApplicationContext;
4. import
5.     org.springframework.context.support.ClassPathXmlApplicationContext;
6. import com.neo.spring.service.CustomerService;
7.
8. public class Client {
9.    private static ApplicationContext context = new
10.    ClassPathXmlApplicationContext(
11.          "com/neo/spring/config/applicationContext.xml");
12.
13.         public static void main(String[] args) {
14.               CustomerService service = (CustomerService)
15.                                         context.getBean("csImpl");
16.               service.printName();
17.               System.out.println();
18.               service.printUrl();
19.               System.out.println();
20.               try {
21.                     service.printException();
22.               } catch (Exception e) {
23.                     System.out.println("Execption raised with
24.                                         message : "+ e.getMessage());
25.               }
26.         }
27.   }
```

AfterReturningAdvice

After returning advice runs when a matched method execution completes normally. It is declared inside an `<aop:aspect>` in the same way as before advice. For example:

It is possible to get hold of the return value within the advice body. Use the "returning" attribute to specify the name of the parameter to which the return value should be passed:
Then method must declare a parameter named `retVal`. The type of this parameter constrains matching in the same way as described for @AfterReturning.



CustomerService.java

```
1. package com.neo.spring.service;
2. public interface CustomerService {
3.          String printName();
4.          String printUrl();
5.          void printException();
6. }
```

CustomerServiceImpl.java

```
1. package com.neo.spring.service;
2.
3. public class CustomerServiceImpl implements CustomerService {
4.          private String name;
5.          private String url;
6.
7.          public void setName(String name) {
8.                this.name = name;
9.          }
10.
11.   public void setUrl(String url) {
12.         this.url = url;
13.   }
14.
15.   @Override
16.   public String printName() {
17.         System.out.println("Business Method: printName() =>"+name);
18.         return name;
19.   }
20.
21.   @Override
22.   public String printUrl() {
23.         System.out.println("Business Method: printUrl() =>" + url);
24.         return url;
```

```
25.   }
26.
27.   @Override
28.   public void printException() {
29.         throw new IllegalArgumentException("Custom Exception");
30.   }
31.   }
```

LoggingAfterReturningAdvice.java

```
1. package com.neo.spring.advice;
2. public class LoggingAfterReturningAdvice {
3.      public void myAfterReturning() {
4.        System.out.println("LoggingAfterReturningAdvice.myAfterReturning");
5.    }
6. }
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2.
3. <beans xmlns="http://www.springframework.org/schema/beans"
4.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xmlns:aop="http://www.springframework.org/schema/aop"
6.    xsi:schemaLocation="http://www.springframework.org/schema/beans
7.         http://www.springframework.org/schema/beans/spring-beans.xsd
8.         http://www.springframework.org/schema/aop
9.         http://www.springframework.org/schema/aop/spring-aop.xsd">
10.
11.   <bean id="csImpl"
12.          class="com.neo.spring.service.CustomerServiceImpl"></bean>
13.   <bean id="lareturn"
14.   class="com.neo.spring.advice.LoggingAfterReturningAdvice"></bean>
15.
16.   <aop:config>
17.
18.    <aop:pointcut id="csPc"
19.         expression="within(com.neo.spring.service.CustomerServiceImpl)"/>
20.    <aop:aspect ref="lareturn">
21.     <aop:after-returning method="myAfterReturning" pointcut-ref="csPc" />
22.    </aop:aspect>
23.
24.   <!-
25.   <aop:aspect ref="lareturn">
26.    <aop:after-returning method="myAfterReturning"
27.          pointcut="within(com.neo.spring.service.CustomerServiceImpl)"/>
28.    </aop:aspect>
29.   -->
30.
31.   </aop:config>
32.
33.   </beans>
```

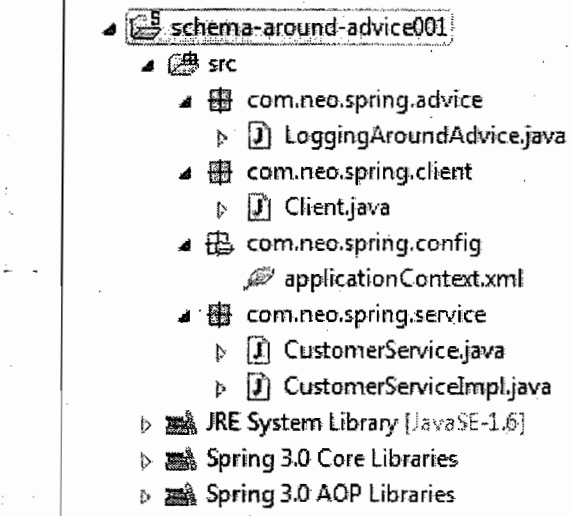
Client.java

```
1. package com.neo.spring.client;
2.
```

```
3. import org.springframework.context.ApplicationContext;
4. import org.springframework.context.support.ClassPathXmlApplicationContext;
5. import com.neo.spring.service.CustomerService;
6.
7. public class Client {
8.    private static ApplicationContext context = new
9.                  ClassPathXmlApplicationContext(
10.                     "com/neo/spring/config/applicationContext.xml");
11.
12.         public static void main(String[] args) {
13.               CustomerService service = (CustomerService)
14.                                          context.getBean("csImpl");
15.               service.printName();
16.               System.out.println();
17.               service.printUrl();
18.               System.out.println();
19.               try {
20.                     service.printException();
21.               } catch (Exception e) {
22.                     System.out.println("Execption raised with message : "
23.                                 + e.getMessage());
24.               }
25.         }
26.   }
```

AroundAdvice

Around advice runs "around" a matched method execution. It has the opportunity to do work both before and after the method executes, and to determine when, how, and even if, the method actually gets to execute at all. Around advice is often used if you need to share state before and after a method execution in a thread-safe manner (starting and stopping a timer for example). Always use the least powerful form of advice that meets your requirements; don't use around advice if simple before advice would do. Around advice is declared using the `aop:around` element. The first parameter of the advice method must be of type `ProceedingJoinPoint`. Within the body of the advice, calling `proceed()` on the `ProceedingJoinPoint` causes the underlying method to execute. The `proceed` method may also be calling passing in an `Object[]` - the values in the array will be used as the arguments to the method execution when it proceeds.



CustomerService.java

```
1. package com.neo.spring.service;
2. public interface CustomerService {
3.          String printName();
4.          String printUrl();
5.          void printException();
6. }
```

CustomerServiceImpl.java

```
1. package com.neo.spring.service;
2.
3. public class CustomerServiceImpl implements CustomerService {
4.          private String name;
5.          private String url;
6.
7.          public void setName(String name) {
8.                this.name = name;
9.          }
10.
11.         public void setUrl(String url) {
12.               this.url = url;
13.         }
14.
15.         @Override
16.         public String printName() {
17.               System.out.println("Business Method: printName() =>"+name);
18.               return name;
19.         }
20.
21.         @Override
22.         public String printUrl() {
23.               System.out.println("Business Method: printUrl() =>" + url);
24.               return url;
25.         }
26.
27.         @Override
28.         public void printException() {
29.               throw new IllegalArgumentException("Custom Exception");
30.         }
31.   }
```

LoggingAroundAdvice.java

```
1. package com.neo.spring.advice;
2.
3. import org.aspectj.lang.ProceedingJoinPoint;
4.
5. public class LoggingAroundAdvice {
6.       public void myAround(ProceedingJoinPoint joinPoint) {
7.          System.out.println("LoggingAroundAdvice.before");
8.          try {
9.                joinPoint.proceed();
10.               } catch (Throwable e) {
11.                     System.out.println("Exception raised :"+
12.                                                   e.getMessage());
```

```
13.               }
14.               System.out.println("LoggingAroundAdvice.after");
15.         }
16.   }
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2.
3. <beans xmlns="http://www.springframework.org/schema/beans"
4.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xmlns:aop="http://www.springframework.org/schema/aop"
6.     xsi:schemaLocation="http://www.springframework.org/schema/beans
7.         http://www.springframework.org/schema/beans/spring-beans.xsd
8.         http://www.springframework.org/schema/aop
9.         http://www.springframework.org/schema/aop/spring-aop.xsd">
10.    <bean id="csImpl"
11.                class="com.neo.spring.service.CustomerServiceImpl"></bean>
12.    <bean id="laround"
13.                class="com.neo.spring.advice.LoggingAroundAdvice"></bean>
14.
15.    <aop:config>
16.      <aop:pointcut id="csPc"
17.          expression="within(com.neo.spring.service.CustomerServiceImpl)"/>
18.
19.      <aop:aspect ref="laround">
20.          <aop:around method="myAround" pointcut-ref="csPc" />
21.      </aop:aspect>
22.
23.      <!-
24.      <aop:aspect ref="laround">
25.          <aop:around method="myAround"
26.            pointcut="within(com.neo.spring.service.CustomerServiceImpl)"/>
27.              </aop:aspect>
28.
29.      -->
30.    </aop:config>
31.    </beans>
```

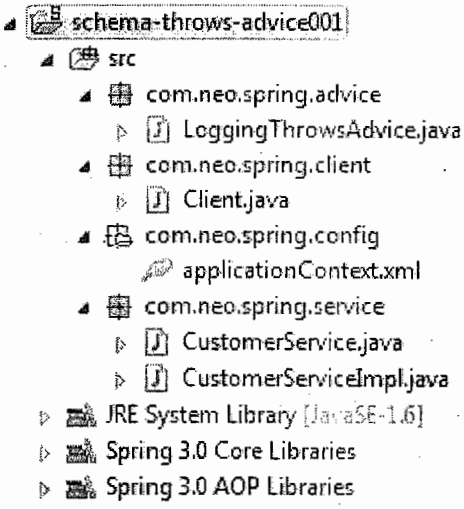
Client.java

```
1. package com.neo.spring.client;
2.
3. import org.springframework.context.ApplicationContext;
4. import org.springframework.context.support.ClassPathXmlApplicationContext;
5. import com.neo.spring.service.CustomerService;
6.
7. public class Client {
8.     private static ApplicationContext context = new
9.               ClassPathXmlApplicationContext(
10.                    "com/neo/spring/config/applicationContext.xml");
11.
12.         public static void main(String[] args) {
13.               CustomerService service = (CustomerService)
14.                                         context.getBean("csImpl");
15.               service.printName();
16.               System.out.println();
```

```
17.                 service.printUrl();
18.                 System.out.println();
19.                 try {
20.                       service.printException();
21.                 } catch (Exception e) {
22.                       System.out.println("Execption raised with message : "
23.                                   + e.getMessage());
24.                 }
25.           }
26.     }
```

AfterThrowingAdvice

After throwing advice executes when a matched method execution exits by throwing an exception. It is declared inside an <aop:aspect> using the after-throwing element:
It is possible to get hold of the thrown exception within the advice body. Use the throwing attribute to specify the name of the parameter to which the exception should be passed.



CustomerService.java

```
1. package com.neo.spring.service;
2. public interface CustomerService {
3.           String printName();
4.           String printUrl();
5.           void printException();
6. }
```

CustomerServiceImpl.java

```
1. package com.neo.spring.service;
2. public class CustomerServiceImpl implements CustomerService {
3.           private String name;
4.           private String url;
5.
6.           public void setName(String name) {
7.                 this.name = name;
8.           }
9.
```

```
10.         public void setUrl(String url) {
11.               this.url = url;
12.         }
13.
14.         @Override
15.         public String printName() {
16.               System.out.println("Business Method: printName() =>"+name);
17.               return name;
18.         }
19.
20.         @Override
21.         public String printUrl() {
22.               System.out.println("Business Method: printUrl() =>" + url);
23.               return url;
24.         }
25.
26.         @Override
27.         public void printException() {
28.               throw new IllegalArgumentException("Custom Exception");
29.         }
30.   }
```

LoggingThrowsAdvice.java

```
1. package com.neo.spring.advice;
2. public class LoggingThrowsAdvice {
3.       public void myAfterThrowing() {
4.          System.out.println("LoggingThrowsAdvice.myAfterThrowing");
5.    }
6. }
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2.
3. <beans xmlns="http://www.springframework.org/schema/beans"
4.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xmlns:aop="http://www.springframework.org/schema/aop"
6.    xsi:schemaLocation="http://www.springframework.org/schema/beans
7.         http://www.springframework.org/schema/beans/spring-beans.xsd
8.         http://www.springframework.org/schema/aop
9.         http://www.springframework.org/schema/aop/spring-aop.xsd">
10.
11.   <bean id="csImpl"
12.               class="com.neo.spring.service.CustomerServiceImpl"></bean>
13.   <bean id="lta"
14.                class="com.neo.spring.advice.LoggingThrowsAdvice"></bean>
15.
16.   <aop:config>
17.   <!--
18.     <aop:pointcut id="csPc"
19.         expression="within(com.neo.spring.service.CustomerServiceImpl)"/>
20.
21.     <aop:aspect ref="lta">
22.        <aop:after-throwing method="myAfterThrowing" pointcut-ref="csPc"/>
23.     </aop:aspect>
24.   -->
```

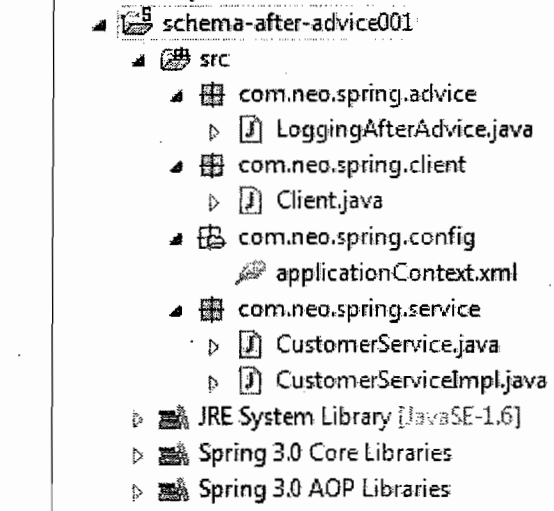
```
25.
26.     <aop:aspect  ref="lta">
27.         <aop:after-throwing method="myAfterThrowing"
28.         pointcut="within(com.neo.spring.service.CustomerServiceImpl)" />
29.     </aop:aspect>
30.   </aop:config>
31.   </beans>
```

Client.java

```
1. package com.neo.spring.client;
2. import org.springframework.context.ApplicationContext;
3. import org.springframework.context.support.ClassPathXmlApplicationContext;
4. import com.neo.spring.service.CustomerService;
5.
6. public class Client {
7.    private static ApplicationContext context = new
8.             ClassPathXmlApplicationContext(
9.                "com/neo/spring/config/applicationContext.xml");
10.
11.         public static void main(String[] args) {
12.               CustomerService service = (CustomerService)
13.                                          context.getBean("csImpl");
14.               service.printName();
15.               System.out.println();
16.               service.printUrl();
17.               System.out.println();
18.               try {
19.                     service.printException();
20.               } catch (Exception e) {
21.                     System.out.println("Execption raised with message : "
22.                                 + e.getMessage());
23.               }
24.         }
25.   }
```

AfterAdvice

After (finally) advice runs however a matched method execution exits. It is declared using the `after` element:



CustomerService.java

```
1. package com.neo.spring.service;
2. public interface CustomerService {
3.          String printName();
4.          String printUrl();
5.          void printException();
6. }
```

CustomerServiceImpl.java

```
1. package com.neo.spring.service;
2.
3. public class CustomerServiceImpl implements CustomerService {
4.          private String name;
5.          private String url;
6.
7.          public void setName(String name) {
8.                this.name = name;
9.          }
10.
11.         public void setUrl(String url) {
12.               this.url = url;
13.         }
14.
15.         @Override
16.         public String printName() {
17.               System.out.println("Business Method: printName() =>"+name);
18.               return name;
19.         }
20.
21.         @Override
22.         public String printUrl() {
23.               System.out.println("Business Method: printUrl() =>" + url);
24.               return url;
25.         }
26.
27.         @Override
28.         public void printException() {
29.               throw new IllegalArgumentException("Custom Exception");
30.         }
31.   }
```

LoggingAfterAdvice.java

```
1. package com.neo.spring.advice;
2. public class LoggingAfterAdvice {
3.       public void myAfter() {
4.          System.out.println("LoggingAfterAdvice.after");
5.    }
6. }
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2.
3. <beans xmlns="http://www.springframework.org/schema/beans"
4.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:aop="http://www.springframework.org/schema/aop"
```

```
5.    xsi:schemaLocation="http://www.springframework.org/schema/beans
6.         http://www.springframework.org/schema/beans/spring-beans.xsd
7.         http://www.springframework.org/schema/aop
8.         http://www.springframework.org/schema/aop/spring-aop.xsd">
9.
10.   <bean id="csImpl"
11.              class="com.neo.spring.service.CustomerServiceImpl"></bean>
12.   <bean id="lafter"
13.                  class="com.neo.spring.advice.LoggingAfterAdvice"></bean>
14.
15.   <aop:config>
16.     <aop:pointcut id="csPc"
17.         expression="within(com.neo.spring.service.CustomerServiceImpl)"/>
18.
19.     <aop:aspect ref="lafter">
20.            <aop:after method="myAfter" pointcut-ref="csPc" />
21.     </aop:aspect>
22.
23.   <!--
24.     <aop:aspect  ref="lafter">
25.         <aop:after method="myAfter"
26.         pointcut="within(com.neo.spring.service.CustomerServiceImpl)" />
27.     </aop:aspect>
28.   -->
29.
30.   </aop:config>
31.   </beans>
```

Client.java

```
1. package com.neo.spring.client;
2.
3. import org.springframework.context.ApplicationContext;
4. import org.springframework.context.support.ClassPathXmlApplicationContext;
5. import com.neo.spring.service.CustomerService;
6.
7. public class Client {
8.    private static ApplicationContext context = new
9.            ClassPathXmlApplicationContext(
10.                    "com/neo/spring/config/applicationContext.xml");
11.
12.         public static void main(String[] args) {
13.               CustomerService service = (CustomerService)
14.                                          context.getBean("csImpl");
15.               service.printName();
16.               System.out.println();
17.               service.printUrl();
18.               System.out.println();
19.               try {
20.                     service.printException();
21.               } catch (Exception e) {
22.                     System.out.println("Execption raised with message : "
23.                                 + e.getMessage());
24.               }
25.         }
26.   }
```

NOTE: We can define, all the advises in single pojo class, and we can apply them to any business object.

LoggingAdvice.java

```
1. package com.neo.spring.advice;
2.
3. import org.aspectj.lang.ProceedingJoinPoint;
4.
5. public class LoggingAdvice {
6.     public void myBefore() {
7.           System.out.println("LoggingAdvice.myBefore()");
8.     }
9.
10.    public void myAfterReturning(Object retVal) {
11.
12.        System.out.println("LoggingAdvice.myAfterReturning():"+retVal);
13.    }
14.
15.    public void myThrowing(Throwable ex) {
16.
17.        System.out.println("LoggingAdvice.myThrowing():"+ex.getMessage());
18.    }
19.
20.    public void myAfter() {
21.     System.out.println("LoggingAdvice.myAfter(){equal to finally block}");
22.    }
23.
24.    public Object myAround(ProceedingJoinPoint joinPoint){
25.          System.out.println("LoggingAdvise.before");
26.          Object retValue=null;
27.          try {
28.                retValue = joinPoint.proceed();
29.          } catch (Throwable e) {
30.
31.                System.out.println("LoggingAdvise.afterThrowing");
32.          }finally{
33.                System.out.println("LoggingAdvise.after");
34.          }
35.
36.          System.out.println("LoggingAdvise.afterReturning");
37.
38.          return retValue;
39.          }
40.
41.    }
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:aop="http://www.springframework.org/schema/aop"
5.     xsi:schemaLocation="http://www.springframework.org/schema/beans
6.          http://www.springframework.org/schema/beans/spring-beans.xsd
7.          http://www.springframework.org/schema/aop
```

```
8.          http://www.springframework.org/schema/aop/spring-aop.xsd">
9.
10.   <bean id="csImpl" class="com.neo.spring.service.CustomerServiceImpl" >
11.               <property name="name" value="sekhar" ></property>
12.               <property name="url" value="sekharit.com" ></property>
13.   </bean>
14.
15.   <bean id="lba" class="com.neo.spring.advice.LoggingAdvice" ></bean>
16.
17.   <aop:config>
18.     <aop:pointcut id="csPC"
19.       expression="within(com.neo.spring.service.CustomerServiceImpl)" />
20.     <aop:pointcut id="servicePackPC"
21.              expression="execution(* com.neo.spring.service.*.*(..))" />
22.
23.     <aop:aspect ref="lba" >
24.         <aop:before method="myBefore" pointcut-ref="csPC" />
25.     </aop:aspect>
26.
27.     <aop:aspect ref="lba" >
28.         <aop:after-returning returning="retVal" method="myAfterReturning"
29.                                                  pointcut-ref="csPC" />
30.     </aop:aspect>
31.
32.     <aop:aspect ref="lba" >
33.         <aop:after-throwing  throwing="ex" method="myThrowing"
34.                                 pointcut="execution(public * *(..))"  />
35.     </aop:aspect>
36.
37.     <aop:aspect ref="lba" >
38.         <aop:after method="myAfter" pointcut-ref="servicePackPC" />
39.     </aop:aspect>
40.
41.     <aop:aspect ref="lba" >
42.         <aop:around method="myAround" pointcut-ref="csPC" />
43.     </aop:aspect>
44.
45.   </aop:config>
46.
47.   </beans>
```

Spring AOP supports the following AspectJ pointcut designators (PCD) for use in pointcut expressions:

- *execution* - for matching method execution join points, this is the primary pointcut designator you will use when working with Spring AOP
- *within* - limits matching to join points within certain types (simply the execution of a method declared within a matching type when using Spring AOP)
- *this* - limits matching to join points (the execution of methods when using Spring AOP) where the bean reference (Spring AOP proxy) is an instance of the given type
- *target* - limits matching to join points (the execution of methods when using Spring AOP) where the target object (application object being proxied) is an instance of the given type
- *args* - limits matching to join points (the execution of methods when using Spring AOP) where the arguments are instances of the given types

Examples

Spring AOP users are likely to use the execution pointcut designator the most often. The format of an execution expression is:

```
execution(modifiers-pattern? ret-type-pattern declaring-type-pattern?
name-pattern(param-pattern)
        throws-pattern?)
```

All parts except the returning type pattern (ret-type-pattern in the snippet above), name pattern, and parameters pattern are optional.

- The returning type pattern determines what the return type of the method must be in order for a join point to be matched. Most frequently we will use `*` as the returning type pattern, which matches any return type. A fully-qualified type name will match only when the method returns the given type.
- The name pattern matches the method name. We can use the `*` wildcard as all or part of a name pattern.
- The parameters pattern is slightly more complex: `()` matches a method that takes no parameters, whereas `(..)` matches any number of parameters (zero or more).
- The pattern `(*)` matches a method taking one parameter of any type, `(*,String)` matches a method taking two parameters, the first can be of any type, the second must be a String.

Some examples of common pointcut expressions are given below.

- the execution of any public method:

```
execution(public * *(..))
```

- the execution of any method with a name beginning with "set":

```
execution(* set*(..))
```

- the execution of any method defined by the `CustomerService` interface:

```
execution(* com.neo.service.CustomerService.*(..))
```

- the execution of any method defined in the service package:

```
execution(* com.neo.service.*.*(..))
```

- the execution of any method defined in the service package or a sub-package:

```
execution(* com.neo.service..*.*(..))
```

- any join point (method execution only in Spring AOP) within the service package:

```
within(com.neo.service.*)
```

- any join point (method execution only in Spring AOP) within the service package or a sub-package:

```
within(com.neo.service..*)
```

- any join point (method execution only in Spring AOP) where the proxy implements the `CustomerService` interface:

```
this(com.neo.service.CustomerService)
```

'this' is more commonly used in a binding form :- see the following section on advice for how to make the proxy object available in the advice body.

- any join point (method execution only in Spring AOP) where the target object implements the `CustomerService` interface:

```
target(com.neo.service.CustomerService)
```

'target' is more commonly used in a binding form :- see the following section on advice for how to make the target object available in the advice body.

- any join point (method execution only in Spring AOP) which takes a single parameter, and where the argument passed at runtime is `Serializable`:

```
args(java.io.Serializable)
```

*'args' is more commonly used in a binding form :- see the following section on * for how to make the method arguments available in the advice body.*

Note that the pointcut given in this example is different to `execution(* *(java.io.Serializable))`: the args version matches if the argument passed at runtime is Serializable, the execution version matches if the method signature declares a single parameter of type `Serializable`.

- any join point (method execution only in Spring AOP) on a Spring bean named '`tradeService`':

```
bean(tradeService)
```

- any join point (method execution only in Spring AOP) on Spring beans having names that match the wildcard expression '`*Service`':

```
bean(*Service)
```

Advantages of Schema-Based AOP:

- To define advice classes we no need to depends on framework given classes and interfaces.
- We can minimize more configurations in spring configuration file.
- We no need to configure ProxyFactoryBean explicitly.
- It have an extra advice called AfterAdvice & it's equal to finally bock.

Annotation Based AOP

In Annotation based AOP we have to use @AspectJ aspects in Spring configuration. We need to enable Spring support for configuring Spring AOP based on @AspectJ aspects, and *autoproxying* beans based on whether or not they are advised by those aspects. By autoproxying we mean that if Spring determines that a bean is advised by one or more aspects, it will automatically generate a proxy for that bean to intercept method invocations and ensure that advice is executed as needed.

If we are using schema, The @AspectJ support is enabled in our spring configuration as follows:

```
<aop:aspectj-autoproxy/>
```

If we are using DTD, The @AspectJ support is enabled in our spring configuration as follows:

```
<bean
class="org.springframework.aop.aspectj.annotation.AnnotationAwareAspectJAutoProxy
Creator" />
```

And also we need two AspectJ libraries on the classpath of our application: **aspectjweaver.jar** and **aspectjrt.jar**. These libraries are available in the **'lib'** directory of an AspectJ installation (version 1.5.1 or later required), or in the **'lib/aspectj'** directory of the Spring-with-dependencies distribution.

Declaring an aspect

With the @AspectJ support enabled, any bean defined in our application context with a class that is an @AspectJ aspect (has the @Aspect annotation) will be automatically detected by Spring and used to configure Spring AOP.

A regular bean definition in the application context, pointing to a bean class that has the @Aspect annotation:

```
<bean id="ma" class="org.neo.advice.MyAspect">
   <!-- configure properties of aspect here as normal -->
</bean>
```

And the **MyAspect** class definition, annotated with **org.aspectj.lang.annotation.Aspect** annotation;

```
package org.xyz;
import org.aspectj.lang.annotation.Aspect;

@Aspect
public class MyAspect {

}
```

Aspects (classes annotated with **@Aspect**) may have methods and fields just like any other class. They may also contain pointcut, advice, and introduction (inter-type) declarations.

Advising aspects

In Spring AOP, it is *not* possible to have aspects themselves be the target of advice from other aspects. The *@Aspect* annotation on a class marks it as an aspect, and hence excludes it from auto-proxying.

Declaring a pointcut

A pointcut declaration has two parts: a signature comprising a name and any parameters, and a pointcut expression that determines *exactly* which method executions we are interested in. In the @AspectJ annotation-style of AOP, a pointcut signature is provided by a regular method definition, and the pointcut expression is indicated using the `@Pointcut` annotation (the method serving as the pointcut signature *must* have a `void` return type).

An example will help make this distinction between a pointcut signature and a pointcut expression clear. The following example defines a pointcut named `'printName'` that will match the execution of any metho named `'Name'`:

```
@Pointcut("execution(* Name(..))")// the pointcut expression
private void printName() {}// the pointcut signature
```

The pointcut expression that forms the value of the `@Pointcut` annotation is a regular AspectJ 5 pointcut expression.

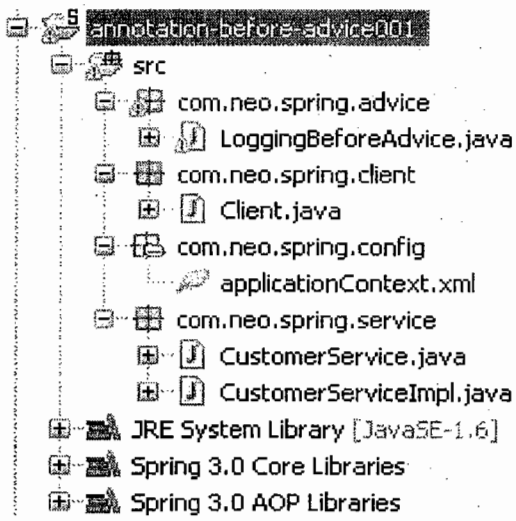
Declaring advice

Advice is associated with a pointcut expression, and runs before, after, or around method executions matched by the pointcut. The pointcut expression may be either a simple reference to a named pointcut, or a pointcut expression declared in place.

BeforeAdvice

Before advice is declared in an aspect using the `@Before` annotation:

We have an example on Before Advice:



CustomerService.java

```
1. package com.neo.spring.service;
2. public interface CustomerService {
3.          String printName();
4.          String printUrl();
5.          void printException();
6. }
```

CustomerServiceImpl.java

```
1. package com.neo.spring.service;
2.
3. public class CustomerServiceImpl implements CustomerService {
4.    private String name;
```

```
5.    private String url;
6.
7.    public void setName(String name) {
8.          this.name = name;
9.    }
10.
11.   public void setUrl(String url) {
12.               this.url = url;
13.   }
14.
15.   @Override
16.   public String printName() {
17.         System.out.println("Business Method: printName() =>" + name);
18.         return name;
19.   }
20.
21.   @Override
22.   public String printUrl() {
23.         System.out.println("Business Method: printUrl() =>" + url);
24.         return url;
25.   }
26.
27.   @Override
28.   public void printException() {
29.         throw new IllegalArgumentException("Custom Exception");
30.    }
31.   }
```

LoggingBeforeAdvice.java

```
1. package com.neo.spring.advice;
2.
3. import org.aspectj.lang.annotation.Aspect;
4. import org.aspectj.lang.annotation.Before;
5.
6. @Aspect
7. public class LoggingBeforeAdvice {
8.     @Before("within(com.neo.spring.service.CustomerServiceImpl)")
9.    public void myBefore() {
10.               System.out.println("LoggingBeforeAdvice.myBefore");
11.         }
12.   }
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.    xmlns:aop="http://www.springframework.org/schema/aop"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5.    xmlns:p="http://www.springframework.org/schema/p"
6.    xsi:schemaLocation="http://www.springframework.org/schema/beans
7. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
8. http://www.springframework.org/schema/aop
9.         http://www.springframework.org/schema/aop/spring-aop.xsd">
10.
11.
12.   <bean id="csImpl"
```

```
13.                 class="com.neo.spring.service.CustomerServiceImpl">
14.      <property name="name" value="somasekhar" />
15.      <property name="url" value="http://neo.com" />
16.   </bean>
17.
18.   <bean id="lba"class="com.neo.spring.advice.LoggingBeforeAdvice"/>
19.
20.    <aop:aspectj-autoproxy />
21.
22.   </beans>
```

Client.java

```
1. package com.neo.spring.client;
2.
3. import org.springframework.context.ApplicationContext;
4. import org.springframework.context.support.ClassPathXmlApplicationContext;
5. import com.neo.spring.service.CustomerService;
6.
7. public class Client {
8.    private static ApplicationContext context = new
9.               ClassPathXmlApplicationContext(
10.                    "com/neo/spring/config/applicationContext.xml");
11.
12.   public static void main(String[] args) {
13.   CustomerService service = (CustomerService)
14.                                          context.getBean("csImpl");
15.
16.         service.printName();
17.         System.out.println();
18.         service.printUrl();
19.         System.out.println();
20.         try {
21.             service.printException();
22.         } catch (Exception e) {
23.               System.out.println("Execption raised with message : "
24.                                  + e.getMessage());
25.               }
26.         }
27.   }
```

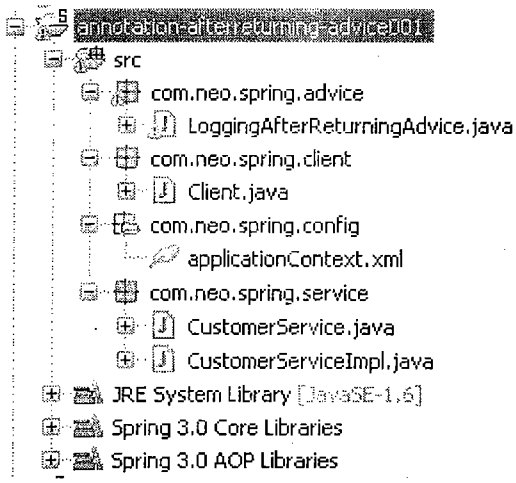
AfterReturningAdvice

After returning advice runs when a matched method execution returns normally. It is declared using the `@AfterReturning` annotation:

Sometimes we need access in the advice body to the actual value that was returned. We can use the form of `@AfterReturning` with `returning` attribute that binds the return value.

The name used in the `returning` attribute must correspond to the name of a parameter in the advice method. When a method execution returns, the return value will be passed to the advice method as the corresponding argument value. A `returning` clause also restricts matching to only those method executions that return a value of the specified type (`Object` in this case, which will match any return value).

Please note that it is *not* possible to return a totally different reference when using after-returning advice.



CustomerService.java

```
1. package com.neo.spring.service;
2. public interface CustomerService {
3.          String printName();
4.          String printUrl();
5.          void printException();
6. }
```

CustomerServiceImpl.java

```
1. package com.neo.spring.service;
2.
3. public class CustomerServiceImpl implements CustomerService {
4.      private String name;
5.      private String url;
6.
7.    public void setName(String name) {
8.          this.name = name;
9.    }
10.
11.   public void setUrl(String url) {
12.             this.url = url;
13.   }
14.
15.   @Override
16.   public String printName() {
17.         System.out.println("Business Method: printName() =>" + name);
18.         return name;
19.   }
20.
21.   @Override
22.   public String printUrl() {
23.         System.out.println("Business Method: printUrl() =>" + url);
24.         return url;
25.   }
26.
27.   @Override
28.   public void printException() {
29.         throw new IllegalArgumentException("Custom Exception");
```

```
30.    }
31.   }
```

LoggingAfterReturningAdvice.java

```
1. package com.neo.spring.advice;
2.
3. import org.aspectj.lang.annotation.AfterReturning;
4. import org.aspectj.lang.annotation.Aspect;
5.
6. @Aspect
7. public class LoggingAfterReturningAdvice {
8.
9. /*
10.   @AfterReturning("within(com.neo.spring.service.CustomerServiceImpl)")
11.   public void myAfterReturning(){
12.
13.       System.out.println("LoggingAfterReturningAdvice.myAfterReturning");
14.   }
15.   */
16.
17.   @AfterReturning(pointcut="within(com.neo.spring.service.
18.                               CustomerServiceImpl)", returning="retValue")
19.   public void myAfterReturning(Object retValue){
20.
21.        System.out.println("LoggingAfterReturningAdvice.myAfterReturning:"
22.                                                              +retValue);
23.         }
24.   }
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:aop="http://www.springframework.org/schema/aop"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5.     xmlns:p="http://www.springframework.org/schema/p"
6.     xsi:schemaLocation="http://www.springframework.org/schema/beans
7. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
8. http://www.springframework.org/schema/aop
9.         http://www.springframework.org/schema/aop/spring-aop.xsd">
10.
11.   <bean id="csImpl" class="com.neo.spring.service.CustomerServiceImpl">
        <property name="name" value="somasekhar" />
12.     <property name="url" value="http://neo.com" />
13.   </bean>
14.
15.   <bean id="lar"
16.             class="com.neo.spring.advice.LoggingAfterReturningAdvice" />
17.
18.   <aop:aspectj-autoproxy />
19.
20.   </beans>
```

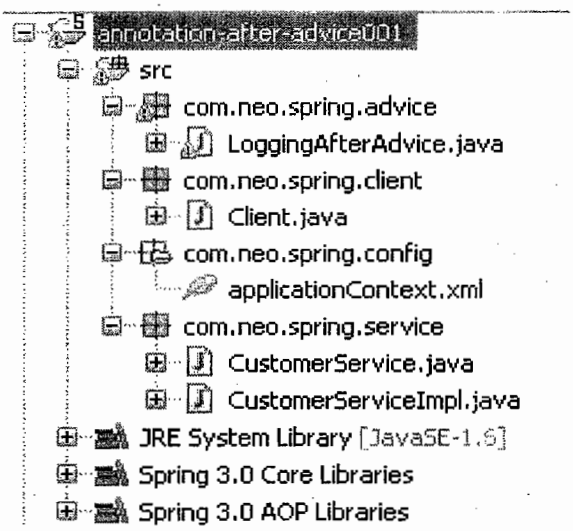
Client.java

```
1. package com.neo.spring.client;
2.
```

```
3. import org.springframework.context.ApplicationContext;
4. import org.springframework.context.support.ClassPathXmlApplicationContext;
5. import com.neo.spring.service.CustomerService;
6.
7. public class Client {
8.      private static ApplicationContext context = new
9.                ClassPathXmlApplicationContext(
10.                     "com/neo/spring/config/applicationContext.xml");
11.
12.   public static void main(String[] args) {
13.   CustomerService service = (CustomerService)
14.                                          context.getBean("csImpl");
15.
16.         service.printName();
17.         System.out.println();
18.         service.printUrl();
19.         System.out.println();
20.         try {
21.             service.printException();
22.         } catch (Exception e) {
23.               System.out.println("Execption raised with message : "
24.                                  + e.getMessage());
25.               }
26.         }
27.   }
```

AfterAdvice

After (finally) advice runs however a matched method execution exits. It is declared using the `@After` annotation. After advice must be prepared to handle both normal and exception return conditions. It is typically used for releasing resources, etc.



CustomerService.java

```
1. package com.neo.spring.service;
2. public interface CustomerService {
3.          String printName();
4.          String printUrl();
```

```
5.          void printException();
6. }
```

CustomerServiceImpl.java

```
1. package com.neo.spring.service;
2.
3. public class CustomerServiceImpl implements CustomerService {
4.      private String name;
5.      private String url;
6.
7.    public void setName(String name) {
8.          this.name = name;
9.    }
10.
11.   public void setUrl(String url) {
12.             this.url = url;
13.   }
14.
15.   @Override
16.   public String printName() {
17.         System.out.println("Business Method: printName() =>" + name);
18.         return name;
19.   }
20.
21.   @Override
22.   public String printUrl() {
23.         System.out.println("Business Method: printUrl() =>" + url);
24.         return url;
25.   }
26.
27.   @Override
28.   public void printException() {
29.         throw new IllegalArgumentException("Custom Exception");
30.    }
31.   }
```

LoggingAfterAdvice.java

```
1. package com.neo.spring.advice;
2.
3. import org.aspectj.lang.annotation.After;
4. import org.aspectj.lang.annotation.Aspect;
5.
6. @Aspect
7. public class LoggingAfterAdvice {
8.
9.    @After("execution(* print*(..))")
10.         public void myAfter() {
11.               System.out.println("LoggingAfterAdvice.myAfter");
12.         }
13.   }
```

applicationContext.xml

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

```
2. <beans xmlns="http://www.springframework.org/schema/beans"
3. xmlns:aop="http://www.springframework.org/schema/aop"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5. xmlns:p="http://www.springframework.org/schema/p"
6. xsi:schemaLocation="http://www.springframework.org/schema/beans
7. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
8. http://www.springframework.org/schema/aop
9. http://www.springframework.org/schema/aop/spring-aop.xsd">
10.
11.    <bean id="csImpl" class="com.neo.spring.service.CustomerServiceImpl">
12.            <property name="name" value="somasekhar" />
13.            <property name="url" value="http://neo.com" />
14.    </bean>
15.
16.    <bean id="laa" class="com.neo.spring.advice.LoggingAfterAdvice" />
17.
18.    <aop:aspectj-autoproxy />
19.
20.    </beans>
```

Client.java

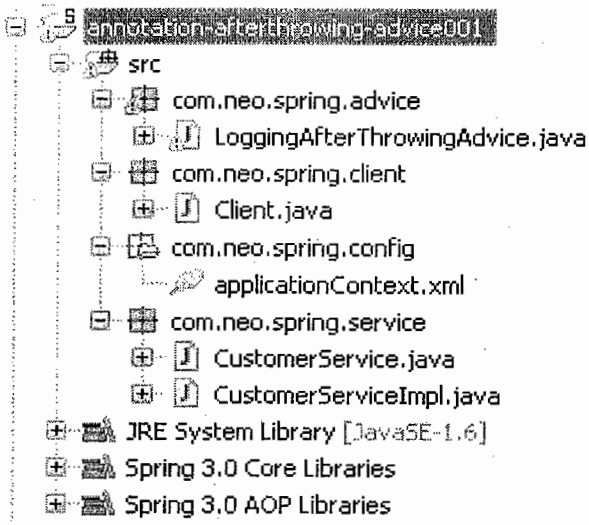
```
1. package com.neo.spring.client;
2.
3. import org.springframework.context.ApplicationContext;
4. import org.springframework.context.support.ClassPathXmlApplicationContext;
5. import com.neo.spring.service.CustomerService;
6.
7. public class Client {
8.      private static ApplicationContext context = new
9.                ClassPathXmlApplicationContext(
10.                     "com/neo/spring/config/applicationContext.xml");
11.
12.   public static void main(String[] args) {
13.   CustomerService service = (CustomerService)
14.                                          context.getBean("csImpl");
15.
16.         service.printName();
17.         System.out.println();
18.         service.printUrl();
19.         System.out.println();
20.         try {
21.             service.printException();
22.         } catch (Exception e) {
23.               System.out.println("Execption raised with message : "
24.                                  + e.getMessage());
25.               }
26.         }
27.   }
```

AfterThrowingAdvice

After throwing advice runs when a matched method execution exits by throwing an exception. It is declared using the `@AfterThrowing` annotation:

Often we want the advice to run only when exceptions of a given type are thrown, and we also often need access to the thrown exception in the advice body. Use the `throwing` attribute to both restrict matching (if desired, use `Throwable` as the exception type otherwise) and bind the thrown exception to an advice parameter.

The name used in the `throwing` attribute must correspond to the name of a parameter in the advice method. When a method execution exits by throwing an exception, the exception will be passed to the advice method as the corresponding argument value. A `throwing` clause also restricts matching to only those method executions that throw an exception of the specified type (`DataAccessException` in this case).



CustomerService.java

```
1. package com.neo.spring.service;
2. public interface CustomerService {
3.          String printName();
4.          String printUrl();
5.          void printException();
6. }
```

CustomerServiceImpl.java

```
1. package com.neo.spring.service;
2.
3. public class CustomerServiceImpl implements CustomerService {
4.      private String name;
5.      private String url;
6.
7.    public void setName(String name) {
8.          this.name = name;
9.    }
10.
11.   public void setUrl(String url) {
12.             this.url = url;
13.   }
14.
15.   @Override
16.   public String printName() {
17.         System.out.println("Business Method: printName() =>" + name);
```

```
18.         return name;
19.    }
20.
21.    @Override
22.    public String printUrl() {
23.         System.out.println("Business Method: printUrl() =>" + url);
24.         return url;
25.    }
26.
27.    @Override
28.    public void printException() {
29.         throw new IllegalArgumentException("Custom Exception");
30.     }
31.    }
```

LoggingAfterThrowingAdvice.java

```
1. package com.neo.spring.advice;
2.
3. import org.aspectj.lang.annotation.AfterThrowing;
4. import org.aspectj.lang.annotation.Aspect;
5.
6. @Aspect
7. public class LoggingAfterThrowingAdvice {
8.
9.     @AfterThrowing(pointcut="within(com.neo.spring.service.
10.                                  CustomerServiceImpl)", throwing="ex")
11.    public void myAfterThrowing(Exception ex){
12.
13.         System.out.println("LoggingAfterThrowingAdvice.myAfterThrowing:"+
14.                                                          ex.getMessage());
15.    }
16.
17.    /*
18.    @AfterThrowing("within(com.neo.spring.service.CustomerServiceImpl)")
19.    public void myAfterThrowing(){
20.         System.out.println("LoggingBeforeAdvice.myAfterThrowing");
21.     }
22.    */
23.
24.    }
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:aop="http://www.springframework.org/schema/aop"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5.     xmlns:p="http://www.springframework.org/schema/p"
6.     xsi:schemaLocation="http://www.springframework.org/schema/beans
7. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
8. http://www.springframework.org/schema/aop
9.          http://www.springframework.org/schema/aop/spring-aop.xsd">
10.
11.    <bean id="csImpl" class="com.neo.spring.service.CustomerServiceImpl">
12.      <property name="name" value="somasekhar" />
```

```
13.     <property name="url" value="http://neo.com" />
14.   </bean>
15.
16.   <bean id="lat"
17.               class="com.neo.spring.advice.LoggingAfterThrowingAdvice" />
18.
19.     <aop:aspectj-autoproxy />
20.
21.   </beans>
```

Client.java

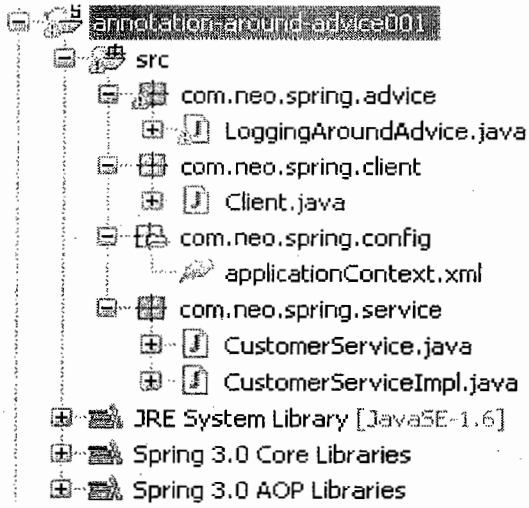
```
1. package com.neo.spring.client;
2.
3. import org.springframework.context.ApplicationContext;
4. import org.springframework.context.support.ClassPathXmlApplicationContext;
5. import com.neo.spring.service.CustomerService;
6.
7. public class Client {
8.      private static ApplicationContext context = new
9.               ClassPathXmlApplicationContext(
10.                    "com/neo/spring/config/applicationContext.xml");
11.
12.   public static void main(String[] args) {
13.   CustomerService service = (CustomerService)
14.                                          context.getBean("csImpl");
15.
16.         service.printName();
17.         System.out.println();
18.         service.printUrl();
19.         System.out.println();
20.         try {
21.             service.printException();
22.         } catch (Exception e) {
23.               System.out.println("Execption raised with message : "
24.                                  + e.getMessage());
25.               }
26.         }
27.   }
```

AroundAdvice

Around advice runs "around" a matched method execution. It has the opportunity to do work both before and after the method executes, and to determine when, how, and even if, the method actually gets to execute at all. Around advice is often used if we need to share state before and after a method execution in a thread-safe manner (starting and stopping a timer for example). Always use the least powerful form of advice that meets your requirements (i.e. don't use around advice if simple before advice would do).

Around advice is declared using the `@Around` annotation. The first parameter of the advice method must be of type `ProceedingJoinPoint`. Within the body of the advice, calling `proceed()` on the `ProceedingJoinPoint` causes the underlying method to execute. The `proceed` method may also be called passing in an `Object[]` - the values in the array will be used as the arguments to the method execution when it proceeds.

The behavior of proceed when called with an Object[] is a little different than the behavior of proceed for around advice compiled by the AspectJ compiler. For around advice written using the traditional AspectJ language, the number of arguments passed to proceed must match the number of arguments passed to the around advice (not the number of arguments taken by the underlying join point), and the value passed to proceed in a given argument position supplants the original value at the join point for the entity the value was bound to (Don't worry if this doesn't make sense right now!). The approach taken by Spring is simpler and a better match to its proxy-based, execution only semantics. You only need to be aware of this difference if you are compiling @AspectJ aspects written for Spring and using proceed with arguments with the AspectJ compiler and weaver. There is a way to write such aspects that is 100% compatible across both Spring AOP and AspectJ, and this is discussed in the following section on advice parameters.



CustomerService.java

```
1. package com.neo.spring.service;
2. public interface CustomerService {
3.          String printName();
4.          String printUrl();
5.          void printException();
6. }
```

CustomerServiceImpl.java

```
1. package com.neo.spring.service;
2.
3. public class CustomerServiceImpl implements CustomerService {
4.      private String name;
5.      private String url;
6.
7.    public void setName(String name) {
8.          this.name = name;
9.    }
10.
11.   public void setUrl(String url) {
12.             this.url = url;
13.   }
14.
15.   @Override
16.   public String printName() {
17.         System.out.println("Business Method: printName() =>" + name);
```

```
18.         return name;
19.    }
20.
21.    @Override
22.    public String printUrl() {
23.         System.out.println("Business Method: printUrl() =>" + url);
24.         return url;
25.    }
26.
27.    @Override
28.    public void printException() {
29.         throw new IllegalArgumentException("Custom Exception");
30.     }
31.    }
```

LoggingAroundAdvice.java

```
1. package com.neo.spring.advice;
2.
3. import org.aspectj.lang.ProceedingJoinPoint;
4. import org.aspectj.lang.annotation.Around;
5. import org.aspectj.lang.annotation.Aspect;
6.
7. @Aspect
8. public class LoggingAroundAdvice {
9.
10.     @Around("within(com.neo.spring.service.CustomerServiceImpl)")
11.     public void myAfterThrowing(ProceedingJoinPoint joinPoint){
12.         System.out.println("LoggingAroundAdvice.myBefore");
13.         try {
14.             Object retValue= joinPoint.proceed();
15.             System.out.println("Return Value : "+retValue);
16.         } catch (Throwable e) {
17.             System.out.println("Exception raised");
18.         }
19.             System.out.println("LoggingAroundAdvice.myAfter");
20.         }
21.    }
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:aop="http://www.springframework.org/schema/aop"
4. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5.     xmlns:p="http://www.springframework.org/schema/p"
6.     xsi:schemaLocation="http://www.springframework.org/schema/beans
7. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
8. http://www.springframework.org/schema/aop
9.         http://www.springframework.org/schema/aop/spring-aop.xsd">
10.
11.    <bean id="csImpl" class="com.neo.spring.service.CustomerServiceImpl">
12.        <property name="name" value="somasekhar" />
13.        <property name="url" value="http://neo.com" />
14.     </bean>
15.
```

```
16.   <bean id="laround" class="com.neo.spring.advice.LoggingAroundAdvice" />
17.
18.   <aop:aspectj-autoproxy />
19.
20.   </beans>
```

Client.java

```
1. package com.neo.spring.client;
2.
3. import org.springframework.context.ApplicationContext;
4. import org.springframework.context.support.ClassPathXmlApplicationContext;
5. import com.neo.spring.service.CustomerService;
6.
7. public class Client {
8.      private static ApplicationContext context = new
9.            ClassPathXmlApplicationContext(
10.                     "com/neo/spring/config/applicationContext.xml");
11.
12.   public static void main(String[] args) {
13.   CustomerService service = (CustomerService)
14.                                          context.getBean("csImpl");
15.
16.         service.printName();
17.         System.out.println();
18.         service.printUrl();
19.         System.out.println();
20.         try {
21.             service.printException();
22.         } catch (Exception e) {
23.               System.out.println("Execption raised with message : "
24.                                  + e.getMessage());
25.               }
26.         }
27.   }
```

Transaction

A **transaction** is an atomic unit of work that either fails or succeeds. There is no such thing as a partial completion of a transaction. Since a transaction can be made up of many steps, each step in the transaction must succeed for the transaction to be successful. If any one part of the transaction fails, then the entire transaction fails. When a transaction fails, the system needs to return to the state that it was in before the transaction was started. This is known as **rollback**. When a transaction fails, then the changes that had been made are said to be "rolled back"

One logical unit of interaction with Database in which SQL statements are grouped and submitted to the DBMS whose changes are either made permanent or undone only as a unit is called as transaction.

Transaction is a set of statements executed on a resource/ resources applying ACID properties.

ACID properties describe the functionalities that have to support for executing/ managing set of statements claimed to be a transaction.

Automicty : Either total statements has to be executed completely or completely undone.

Consistency : The database state in the database after transaction should transform from one consistent state to another consistent state.

Isolation : One transaction should not affect with any other transaction even they are executing on the same resource.

Durability : Transaction will be ended successfully even on system failures.

There are two types of transactions

a. Local transaction
b. Distributed transaction

Local Transaction

- ⇨ Local transaction will be executed on single resource within the same session.
- ⇨ Local transactions are managed by Resource manager.

Distributed transaction

- ⇨ Distributed transactions can be executed on multiple resources across multiple sessions.
- ⇨ Distributed transactions are managed by specially designed external transaction manager.

Every transaction has two boundaries

- o Beginning
- o Ending

⇨ Controlling the boundaries of a transaction is nothing but transaction management.

Transaction management is critical in any form of applications that will interact with the database. The application has to ensure that the data is consistent and the integrity of the data is maintained. There are many popular data access frameworks like **JDBC**, **JPA**, **Hibernate** etc.... And **Spring Framework** provides a seamless way of integrating with these frameworks.

Benefits of Spring Transaction Management

- Very easy to use, does not require any underlying transaction API knowledge
- Transaction management code will be independent of the transaction technology
- Both annotation- and XML-based configuration support
- It does not require to run on a server - no server needed

Spring offers two ways of handling transactions: programmatic and declarative.

Programmatic means we have transaction management code surrounding our business code. That gives us extreme flexibility, but is difficult to maintain and well, boilerplate.

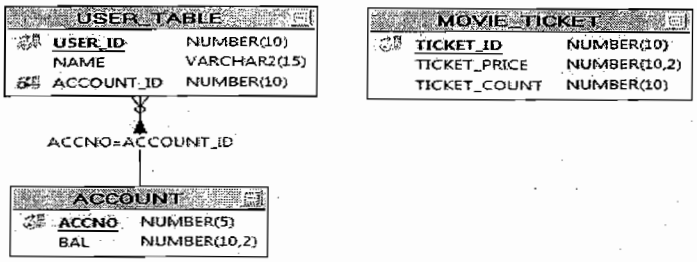
Declarative means we separate transaction management from the business code. We only use annotations or XML based configuration.

- programmatic management is more flexible during development time but less flexible during application life
- declarative management is less flexible during development time but more flexible during application life

Transaction Management without Spring

In this section we will see how an application tends to manage transactions. We will develop an application which will provide a service to **book a movie ticket**. An User can book a Movie ticket in that case the amount will be debited from his/her Account.

Tables and their Relationships:



ACCOUNT

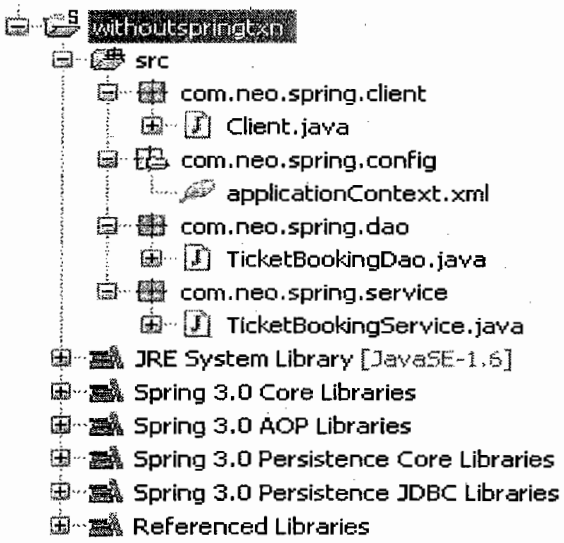
ACCNO	BAL
1001	3000
1002	5000

USER_TABLE

USER_ID	NAME	ACCOUNT_ID
9001	SEKHAR	1001
9002	SOMU	1002

MOVIE_TICKET

TICKET_ID	TICKET_PRICE	TICKET_COUNT
5001	45	300
5002	30	150



TicketBookingDao.java

```
1. package com.neo.spring.dao;
2.
3. import java.sql.Connection;
4. import java.sql.ResultSet;
5. import java.sql.SQLException;
6. import java.sql.Statement;
7.
8. public class TicketBookingDao {
9.          private Connection connection;
10.
11.         public void setConnection(Connection connection) {
12.               this.connection = connection;
13.         }
14.
15.         public int getAccountId(int userId) throws SQLException {
16.               int accountId = 0;
17.               String query = "SELECT ACCOUNT_ID FROM USER_TABLE
18.                                          WHERE USER_ID=" + userId;
19.               Statement statement;
20.                     statement = connection.createStatement();
21.                     ResultSet resultSet =
22.                                       statement.executeQuery(query);
23.                     if (resultSet.next()) {
24.                           accountId = resultSet.getInt(1);
25.                     }
26.               return accountId;
27.         }
28.
29.         public float getPrice(int ticketId) throws SQLException{
30.               float price=0;
31.               String query="SELECT TICKET_PRICE FROM MOVIE_TICKET
32.                                          WHERE TICKET_ID="+ticketId;
33.               Statement statement = connection.createStatement();
34.               ResultSet resultSet =statement.executeQuery(query);
35.                     if(resultSet.next()){
36.                           price= resultSet.getFloat(1);
37.                     }
38.               return price;
39.         }
40.
41.         public void deductAmount(int accountId, double amount)
42.                                                throws SQLException{
43.               String query="UPDATE ACCOUNT SET BAL=BAL-"+amount+"
44.                                          WHERE ACCNO="+accountId;
45.               Statement statement = connection.createStatement();
46.                     statement.executeUpdate(query);
47.         }
48.
49.         public void reduceTicketCount(int ticketId, int
50.                                    noOfTickets) throws SQLException{
51.               String query="UPDATE MOVIE_TICKET SET TICKET_COUNT=
52.            TICKET_COUNT-"+noOfTickets+" WHERE TICKET_ID="+ticketId;
53.               Statement statement = connection.createStatement();
54.                     statement.executeUpdate(query);
```

```
55.         }
56.   }
```

TicketBookingService.java

```
1. package com.neo.spring.service;
2.
3. import java.sql.Connection;
4. import java.sql.SQLException;
5. import javax.sql.DataSource;
6. import com.neo.spring.dao.TicketBookingDao;
7.
8. public class TicketBookingService {
9.    private TicketBookingDao dao;
10.         private DataSource dataSource;
11.
12.         public void setDao(TicketBookingDao dao) {
13.               this.dao = dao;
14.         }
15.
16.         public void setDataSource(DataSource dataSource) {
17.               this.dataSource = dataSource;
18.         }
19.
20.         public void bookTicket(int userId, int ticketId, int
21.                                                totalNoTickets) {
22.               Connection connection = null;
23.
24.               try {
25.                     connection = dataSource.getConnection();
26.                     connection.setAutoCommit(false);
27.                     dao.setConnection(connection);
28.                     int accountId = dao.getAccountId(userId);
29.                     float price = dao.getPrice(ticketId);
30.                     double totalAmount= price*totalNoTickets;
31.                     dao.deductAmount(accountId, totalAmount);
32.                     dao.reduceTicketCount(ticketId,totalNoTickets);
33.
34.                     connection.commit();
35.               } catch (Exception e) {
36.                     try {
37.                           connection.rollback();
38.                     } catch (SQLException e1) {
39.                           e1.printStackTrace();
40.                     }
41.                     System.out.println("...."+e.getMessage());
42.               }
43.         }
44.   }
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:p="http://www.springframework.org/schema/p"
5. xsi:schemaLocation="http://www.springframework.org/schema/beans
```

```
6. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7.
8.      <bean id="ds"
9.        class="org.springframework.jdbc.datasource.DriverManagerDataSource">
10.                 <property name="driverClassName">
11.                     <value>oracle.jdbc.driver.OracleDriver</value>
12.      </property>
13.      <property name="url">
14.          <value>jdbc:oracle:thin:@localhost:1521:server</value>
15.      </property>
16.      <property name="username">
17.          <value>scott</value>
18.      </property>
19.      <property name="password">
20.          <value>tiger</value>
21.      </property>
22.    </bean>
23.
24.    <bean id="bookingDaoRef"
25.                          class="com.neo.spring.dao.TicketBookingDao" />
26.
27.    <bean id="bookingServiceRef"
28.                      class="com.neo.spring.service.TicketBookingService">
29.      <property name="dao" ref="bookingDaoRef" />
30.      <property name="dataSource" ref="ds" />
31.    </bean>
32.
33.    </beans>
```

Client.java

```
1. package com.neo.spring.client;
2.
3. import org.springframework.context.ApplicationContext;
4. import
5.      org.springframework.context.support.ClassPathXmlApplicationContext;
6. import com.neo.spring.service.TicketBookingService;
7.
8. public class Client {
9.       private static ApplicationContext context = new
10.                                    ClassPathXmlApplicationContext(
11.                    "com/neo/spring/config/applicationContext.xml");
12.
13.         public static void main(String[] args) {
14.
15.               TicketBookingService service =
16.         (TicketBookingService)context.getBean("bookingServiceRef");
17.               service.bookTicket(9001, 5001, 5);
18.         System.out.println("Successfully tickets are booked ");
19.         }
20.    }
```

OUTPUT

```
Successfully tickets are booked
```

ACCOUNT

ACCNO	BAL
1001	2775
1002	5000

USER_TABLE

USER_ID	NAME	ACCOUNT_ID
9001	SEKHAR	1001
9002	SOMU	1002

MOVIE_TICKET

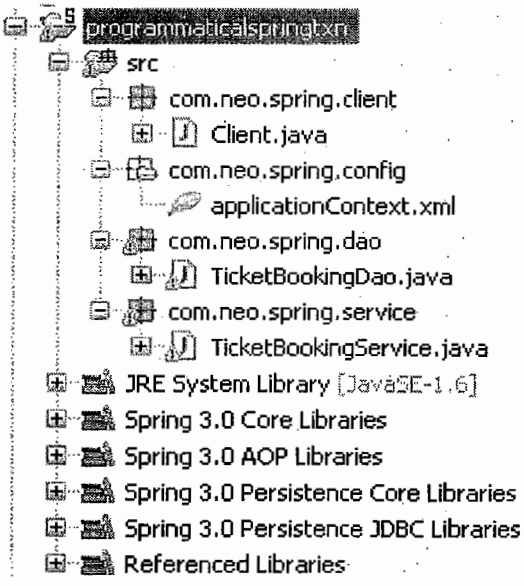
TICKET_ID	TICKET_PRICE	TICKET_COUNT
5001	45	295
5002	30	150

In the previous section, we saw how transaction got maintained by the application, in which case application has to manually do a commit or a rollback in the case of a successful or a failure transactions.

And we are writing connection related code in the service layer. It violates the rule of clear separation of data access logic and business logic.

Programmatic Transaction Management – Transaction Manager

Here, we will see how to manage transactions using Spring framework. Spring provides support for both Programmatic and Declarative mode of transaction management, although most of the applications prefer using Declarative mode of transaction management since the transaction related code is completed shielded from the application code. The advantage in having Programmatic mode of transaction management is that the application will have greater control over the transaction.



TicketBookingDao.java

```
1. package com.neo.spring.dao;
2.
3. import java.sql.Connection;
4. import java.sql.ResultSet;
5. import java.sql.SQLException;
6. import java.sql.Statement;
7. import org.springframework.jdbc.core.support.JdbcDaoSupport;
8.
9.  public class TicketBookingDao extends JdbcDaoSupport {
10.
11.         public int getAccountId(int userId) throws SQLException {
12.               int accountId = 0;
```

```
13.               String query = "SELECT ACCOUNT_ID FROM USER_TABLE
14.                                          WHERE USER_ID="+ userId;
15.               accountId  = getJdbcTemplate().queryForInt(query);
16.               return accountId;
17.         }
18.
19.         public float getPrice(int ticketId) throws SQLException{
20.               float price=0;
21.               String query="SELECT TICKET_PRICE FROM MOVIE_TICKET
22.                                       WHERE TICKET_ID="+ticketId;
23.               price = getJdbcTemplate().queryForObject(query,
24.                                                   Float.class);
25.               return price;
26.         }
27.
28.         public void deductAmount(int accountId, double amount)
29.                                              throws SQLException{
30.               String query="UPDATE ACCOUNT SET BAL=BAL-"+amount+"
31.                                          WHERE ACCNO="+accountId;
32.                     getJdbcTemplate().execute(query);
33.         }
34.
35.         public void reduceTicketCount(int ticketId, int
36.                                    noOfTickets) throws SQLException{
37.               String query="UPDATE MOVIE_TICKET SET TICKET_COUNT=
38.            TICKET_COUNT-"+noOfTickets+" WHERE TICKET_ID="+ticketId;
39.                     getJdbcTemplate().execute(query);
40.         }
41.   }
```

TicketBookingService.java

```
1. package com.neo.spring.service;
2.
3. import java.sql.Connection;
4. import java.sql.SQLException;
5. import javax.sql.DataSource;
6. import org.springframework.transaction.PlatformTransactionManager;
7. import org.springframework.transaction.TransactionDefinition;
8. import org.springframework.transaction.TransactionStatus;
9. import
       org.springframework.transaction.support.DefaultTransactionDefinition;
10.   import com.neo.spring.dao.TicketBookingDao;
11.
12.   public class TicketBookingService {
13.         private TicketBookingDao dao;
14.         private PlatformTransactionManager manager;
15.
16.         public void setManager(PlatformTransactionManager manager){
17.               this.manager = manager;
18.         }
19.
20.         public void setDao(TicketBookingDao dao) {
21.               this.dao = dao;
22.         }
23.
```

```
24.         public void bookTicket(int userId, int ticketId, int
25.                                                   totalNoTickets) {
26.               TransactionDefinition definition = new
27.                                       DefaultTransactionDefinition();
28.               TransactionStatus status =
29.                                    manager.getTransaction(definition);
30.               try {
31.                     int accountId = dao.getAccountId(userId);
32.                     float price = dao.getPrice(ticketId);
33.                     double totalAmount = price * totalNoTickets;
34.                     dao.deductAmount(accountId, totalAmount);
35.                     dao.reduceTicketCount(ticketId, totalNoTickets);
36.                     manager.commit(status);
37.               } catch (Exception e) {
38.                     manager.rollback(status);
39.                     System.out.println("...." + e.getMessage());
40.               }
41.         }
42.   }
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:p="http://www.springframework.org/schema/p"
5.     xsi:schemaLocation="http://www.springframework.org/schema/beans
6. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7.
8.       <bean id="ds"
9.        class="org.springframework.jdbc.datasource.DriverManagerDataSource">
10.        <property name="driverClassName">
11.                      <value>oracle.jdbc.driver.OracleDriver</value>
12.        </property>
13.        <property name="url">
14.          <value>jdbc:oracle:thin:@localhost:1521:server</value>
15.        </property>
16.        <property name="username">
17.          <value>scott</value>
18.        </property>
19.        <property name="password">
20.          <value>tiger</value>
21.        </property>
22.      </bean>
23.
24.      <bean id="bookingDaoRef"
25.                               class="com.neo.spring.dao.TicketBookingDao">
26.        <property name="dataSource" ref="ds" />
27.      </bean>
28.
29.      <bean id="bookingServiceRef"
30.                        class="com.neo.spring.service.TicketBookingService">
31.        <property name="dao" ref="bookingDaoRef" />
32.        <property name="manager" ref="tm" />
33.      </bean>
34.
```

```
35.     <bean id="tm"
36.   class="org.springframework.jdbc.datasource.DataSourceTransaction
37.                                                          Manager">
38.       <property name="dataSource" ref="ds" />
39.     </bean>
40.
41.   </beans>
```

Client.java

```
1. package com.neo.spring.client;
2.
3. import org.springframework.context.ApplicationContext;
4. import
5.     org.springframework.context.support.ClassPathXmlApplicationContext;
6. import com.neo.spring.service.TicketBookingService;
7.
8. public class Client {
9.          private static ApplicationContext context = new
10.                                    ClassPathXmlApplicationContext(
11.                   "com/neo/spring/config/applicationContext.xml");
12.
13.        public static void main(String[] args) {
14.
15.              TicketBookingService service =
16.        (TicketBookingService)context.getBean("bookingServiceRef");
17.              service.bookTicket(9002, 5002, 10);
18.           System.out.println("Successfully tickets are booked ");
19.        }
20.   }
```

OUTPUT

```
Successfully tickets are booked
```

ACCOUNT

ACCNO	BAL
1001	2775
1002	4700

USER_TABLE

USER_ID	NAME	ACCOUNT_ID
9001	SEKHAR	1001
9002	SOMU	1002

MOVIE_TICKET

TICKET_ID	TICKET_PRICE	TICKET_COUNT
5001	45	295
5002	30	150

Schema based Transaction Management

In schema based Transaction management, we configure the transaction logic in configuration file instead of writing in java code.

AOP programming enables the concerns that cross-cut over the application can be separated and can be encapsulated. Since marking the beginning of a transaction, end of a transaction, commiting the database in the case of successful operation and making a rollback for failure operations are really cross-cutting concerns, it is ideal to take AOP based approach for this scenario.

Note that we have declared the proxy bean '`serviceProxy`' which is an instance of 'org.springframework.aop.framework.ProxyFactoryBean'. The target for this proxy will be the **TicketBookingService** object which is specified through 'target' property. And because we want to intercept the transaction related operations, we have specified the interceptor '`txnInterceptor`' through the element 'interceptorNames'.

ACCOUNT

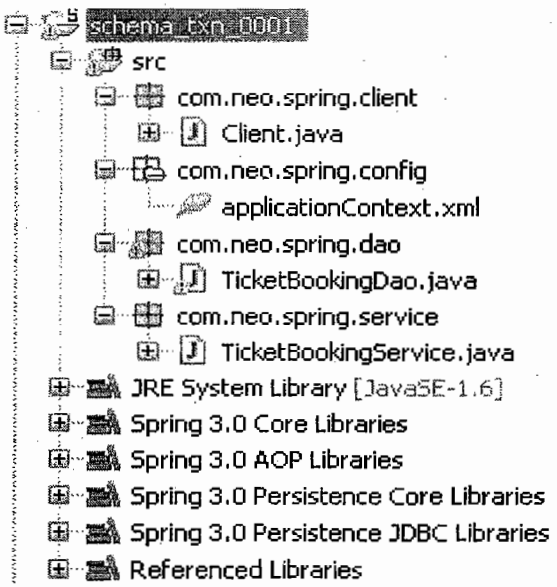
ACCNO	NAME	BAL
1001	SEKHAR	2775
1002	SOMU	4700

USER_TABLE

USER_ID	ACCOUNT_ID
9001	1001
9002	1002

MOVIE_TICKET

TICKET_ID	TICKET_PRICE	TICKET_COUNT
5001	45	295
5002	30	140



TicketBookingDao.java

```
1. package com.neo.spring.dao;
2.
3. import java.sql.Connection;
4. import java.sql.ResultSet;
5. import java.sql.SQLException;
6. import java.sql.Statement;
7. import org.springframework.jdbc.core.support.JdbcDaoSupport;
```

```
8.
9. public class TicketBookingDao extends JdbcDaoSupport {
10.
11.         public int getAccountId(int userId) throws SQLException {
12.               int accountId = 0;
13.               String query = "SELECT ACCOUNT_ID FROM USER_TABLE WHERE
14.                                                          USER_ID="+ userId;
15.               accountId  = getJdbcTemplate().queryForInt(query);
16.               return accountId;
17.         }
18.
19.         public float getPrice(int ticketId) throws SQLException{
20.               float price=0;
21.               String query="SELECT TICKET_PRICE FROM MOVIE_TICKET WHERE
22.                                                      TICKET_ID="+ticketId;
23.               price = getJdbcTemplate().queryForObject(query,
24.                                                            Float.class);
25.               return price;
26.         }
27.
28.         public void deductAmount(int accountId, double amount) throws
29.                                                            SQLException{
30.               String query="UPDATE ACCOUNT SET BAL=BAL-"+amount+" WHERE
31.                                                      ACCNO="+accountId;
32.                     getJdbcTemplate().execute(query);
33.         }
34.
35.         public void reduceTicketCount(int ticketId, int noOfTickets)
36.                                                     throws SQLException{
37.               String query="UPDATE MOVIE_TICKET SET TICKET_COUNT=
38.                  TICKET_COUNT-"+noOfTickets+" WHERE TICKET_ID="+ticketId;
39.                     getJdbcTemplate().execute(query);
40.         }
41.   }
```

TicketBookingService.java

```
1. package com.neo.spring.service;
2.
3. import com.neo.spring.dao.TicketBookingDao;
4.
5. public class TicketBookingService {
6.     private TicketBookingDao dao;
7.
8.     public void setDao(TicketBookingDao dao) {
9.             this.dao = dao;
10.    }
11.
12.     public void bookTicket(int userId, int ticketId, int totalNoTickets){
13.               try {
14.                     int accountId = dao.getAccountId(userId);
15.                     float price = dao.getPrice(ticketId);
16.                     double totalAmount = price * totalNoTickets;
17.                     dao.deductAmount(accountId, totalAmount);
18.                     dao.reduceTicketCount(ticketId, totalNoTickets);
19.               } catch (Exception e) {
```

```
20.                     System.out.println("...." + e.getMessage());
21.                 }
22.         }
23.   }
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.   xmlns:p="http://www.springframework.org/schema/p"
5.     xsi:schemaLocation="http://www.springframework.org/schema/beans
6.   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7.
8.       <bean id="ds"
9.       class="org.springframework.jdbc.datasource.DriverManagerDataSource">
10.          <property name="driverClassName">
11.               <value>oracle.jdbc.driver.OracleDriver</value>
12.          </property>
13.          <property name="url">
14.               <value>jdbc:oracle:thin:@localhost:1521:server</value>
15.          </property>
16.          <property name="username">
17.               <value>scott</value>
18.          </property>
19.          <property name="password">
20.               <value>tiger</value>
21.          </property>
22.       </bean>
23.
24.      <bean id="bookingDaoRef" class="com.neo.spring.dao.TicketBookingDao">
25.          <property name="dataSource" ref="ds" />
26.      </bean>
27.
28.      <bean id="bookingServiceRef"
29.      class="com.neo.spring.service.TicketBookingService">
30.          <property name="dao" ref="bookingDaoRef" />
31.      </bean>
32.
33.      <bean id="tm"
34.    class="org.springframework.jdbc.datasource.DataSourceTransactionManager
35.                                                                        ">
36.          <property name="dataSource" ref="ds" />
37.      </bean>
38.
39.    <!--
40.       <bean id="txnInterceptor"
41.    class="org.springframework.transaction.interceptor.TransactionIntercept
42.                                                                    or">
43.          <property name="transactionManager" ref="tm"></property>
44.          <property name="transactionAttributes">
45.               <props>
46.                    <prop key="bookTicket">PROPAGATION_REQUIRED</prop>
47.               </props>
48.          </property>
49.       </bean>
```

```
50.
51.     <bean id="serviceProxy"
52.       class="org.springframework.aop.framework.ProxyFactoryBean">
53.         <property name="target" ref="bookingServiceRef"></property>
54.         <property name="interceptorNames">
55.               <list>
56.                     <value>txnInterceptor</value>
57.               </list>
58.         </property>
59.     </bean>
60.   -->
61.
62.     <bean id="serviceProxy"
63.     class="org.springframework.transaction.interceptor.TransactionProxy
64.                                                          FactoryBean">
65.         <property name="target" ref="bookingServiceRef" />
66.         <property name="transactionManager" ref="tm" />
67.         <property name="transactionAttributes">
68.               <props>
69.                     <prop key="bookTicket">PROPAGATION_REQUIRED</prop>
70.               </props>
71.         </property>
72.     </bean>
73.
74.   </beans>
```

Client.java

```
1. package com.neo.spring.client;
2.
3. import org.springframework.context.ApplicationContext;
4. import
5.     org.springframework.context.support.ClassPathXmlApplicationContext;
6. import com.neo.spring.service.TicketBookingService;
7.
8. public class Client {
9.         private static ApplicationContext context = new
10.                                     ClassPathXmlApplicationContext(
11.                   "com/neo/spring/config/applicationContext.xml");
12.
13.         public static void main(String[] args) {
14.
15.               TicketBookingService service =
16.         (TicketBookingService)context.getBean("bookingServiceRef");
17.               service.bookTicket(9001, 5001, 5);
18.         System.out.println("Successfully tickets are booked ");
19.         }
20.   }
```

OUTPUT

```
Successfully tickets are booked
```

ACCOUNT

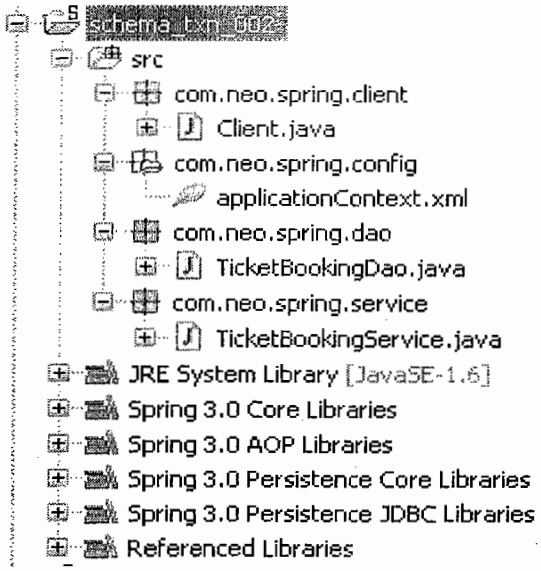
ACCNO	NAME	BAL
1001	SEKHAR	2550
1002	SOMU	4700

USER_TABLE

USER_ID	ACCOUNT_ID
9001	1001
9002	1002

MOVIE_TICKET

TICKET_ID	TICKET_PRICE	TICKET_COUNT
5001	45	290
5002	30	140



TicketBookingDao.java

```
1. package com.neo.spring.dao;
2.
3. import java.sql.SQLException;
4. import org.springframework.jdbc.core.support.JdbcDaoSupport;
5.
6. public class TicketBookingDao extends JdbcDaoSupport {
7.
8.     public int getAccountId(int userId) throws SQLException {
9.          int accountId = 0;
10.         String query = "SELECT ACCOUNT_ID FROM USER_TABLE WHERE USER_ID="
11.                                                              + userId;
12.               accountId  = getJdbcTemplate().queryForInt(query);
13.               return accountId;
14.    }
15.
16.         public float getPrice(int ticketId) throws SQLException{
17.               float price=0;
18.               String query="SELECT TICKET_PRICE FROM MOVIE_TICKET WHERE
19.                                                    TICKET_ID="+ticketId;
20.               price = getJdbcTemplate().queryForObject(query,
21.                                                         Float.class);
22.               return price;
23.         }
24.
25.         public void deductAmount(int accountId, double amount) throws
26.                                                         SQLException{
```

```
27.               String query="UPDATE ACCOUNT SET BAL=BAL-"+amount+" WHERE
28.                                                        ACCNO="+accountId;
29.                     getJdbcTemplate().execute(query);
30.         }
31.
32.         public void reduceTicketCount(int ticketId, int noOfTickets)
33.                                                        throws SQLException{
34.               String query="UPDATE MOVIE_TICKET SET TICKET_COUNT=
35.                  TICKET_COUNT-"+noOfTickets+" WHERE TICKET_ID="+ticketId;
36.                     getJdbcTemplate().execute(query);
37.         }
38.   }
```

TicketBookingService.java

```
1. package com.neo.spring.service;
2.
3. import com.neo.spring.dao.TicketBookingDao;
4.
5. public class TicketBookingService {
6.    private TicketBookingDao dao;
7.
8.    public void setDao(TicketBookingDao dao) {
9.          this.dao = dao;
10.   }
11.   public void x(int userId, int ticketId, int totalNoTickets){
12.         bookTicket(userId, ticketId, totalNoTickets);
13.   }
14.   public void bookTicket(int userId, int ticketId, int totalNoTickets) {
15.         try {
16.               int accountId = dao.getAccountId(userId);
17.               float price = dao.getPrice(ticketId);
18.               double totalAmount = price * totalNoTickets;
19.               dao.deductAmount(accountId, totalAmount);
20.               dao.reduceTicketCount(ticketId, totalNoTickets);
21.         } catch (Exception e) {
22.               System.out.println("...." + e.getMessage());
23.         }
24.      }
25.   }
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:aop="http://www.springframework.org/schema/aop"
5.    xmlns:tx="http://www.springframework.org/schema/tx"
6.    xsi:schemaLocation="
7.        http://www.springframework.org/schema/beans
8. http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
9.        http://www.springframework.org/schema/tx
10.   http://www.springframework.org/schema/tx/spring-tx-2.0.xsd
11.         http://www.springframework.org/schema/aop
12.   http://www.springframework.org/schema/aop/spring-aop-2.0.xsd">
13.
14.      <bean id="ds"
```

```
15.     class="org.springframework.jdbc.datasource.DriverManagerDataSource">
16.        <property name="driverClassName">
17.              <value>oracle.jdbc.driver.OracleDriver</value>
18.        </property>
19.        <property name="url">
20.              <value>jdbc:oracle:thin:@localhost:1521:server</value>
21.        </property>
22.        <property name="username">
23.              <value>scott</value>
24.        </property>
25.        <property name="password">
26.              <value>tiger</value>
27.        </property>
28.     </bean>
29.
30.     <bean id="bookingDaoRef" class="com.neo.spring.dao.TicketBookingDao">
31.        <property name="dataSource" ref="ds" />
32.     </bean>
33.
34.      <bean id="bookingServiceRef"
35.                    class="com.neo.spring.service.TicketBookingService">
36.        <property name="dao" ref="bookingDaoRef" />
37.      </bean>
38.
39.      <bean id="tm"
40.   class="org.springframework.jdbc.datasource.DataSourceTransactionManager
41.                                                                    ">
42.        <property name="dataSource" ref="ds" />
43.      </bean>
44.
45.      <tx:advice id="transactionAdvice" transaction-manager="tm">
46.        <tx:attributes>
47.              <tx:method name="bookTicket" />
48.        </tx:attributes>
49.      </tx:advice>
50.
51.      <aop:config>
52.        <aop:pointcut id="myPointcut"
53.   expression="within(com.neo.spring.service.TicketBookingService)" />
54.        <aop:advisor advice-ref="transactionAdvice" pointcut-
55.                                                    ref="myPointcut" />
56.       </aop:config>
57.
58.   </beans>
```

Client.java

```
1. package com.neo.spring.client;
2.
3. import org.springframework.context.ApplicationContext;
4. import
5.    org.springframework.context.support.ClassPathXmlApplicationContext;
6. import com.neo.spring.service.TicketBookingService;
7.
8. public class Client {
9.    private static ApplicationContext context = new
```

```
10.                                          ClassPathXmlApplicationContext(
11.                      "com/neo/spring/config/applicationContext.xml");
12.
13.         public static void main(String[] args) {
14.
15.               TicketBookingService service =
16.               (TicketBookingService)context.getBean("bookingServiceRef");
17.               service.x(9001, 5001, 5);
18.               System.out.println("Successfully tickets are booked ");
19.         }
20.   }
```

OUTPUT

```
Successfully tickets are booked
```

ACCOUNT

ACCNO	NAME	BAL
1001	SEKHAR	2325
1002	SOMU	4700

USER_TABLE

USER_ID	ACCOUNT_ID
9001	1001
9002	1002

MOVIE_TICKET

TICKET_ID	TICKET_PRICE	TICKET_COUNT
5001	45	285
5002	30	140

The <tx:advice/> definition reads all the methods which are given in <tx:method />. The 'transaction-manager' attribute of the <tx:advice/> tag is set to the name of the PlatformTransactionManager bean that is going to actually *drive* the transactions (in this case the 'tm' bean).

We can actually omit the 'transaction-manager' attribute in the transactional advice (<tx:advice/>) if the bean name of the PlatformTransactionManager that we want to wire in has the name 'transactionManager'. If the PlatformTransactionManager bean that we want to wire in has any other name, then we have to be explicit and use the 'transaction-manager' attribute.

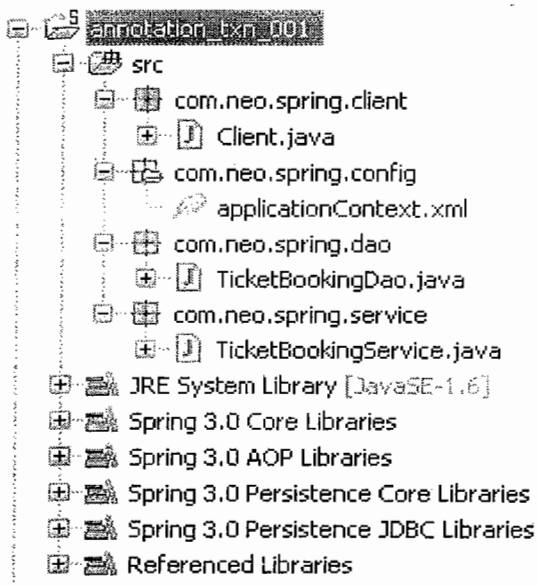
The `<aop:config/>` definition ensures that the transactional advice defined by the ***transactionAdvice*** bean actually executes at the appropriate points in the program. First we define a pointcut that matches the execution of any operation defined in the ***myPointcut***. Then we associate the pointcut with the ***transactionAdvice*** using an advisor. The result indicates that at the execution of a ***myPointcut***, the advice defined by ***transactionAdvice*** will be run.

Annotation based Transaction Management

@Transactional is the annotation we can use on any method of any bean and on the bean itself.
If we apply the annotation on the bean (i.e. on the class level), every public method will be transactional.

Note: Remember that it affects only Spring managed data sources (in our example whatever comes from the entity manager factory). If you get your data source outside of Spring, the @Transactional annotation will have no effect.

We should put this annotation on our business logic methods (service methods), not on DAO methods. Normally a business method will call many DAO methods and those calls make only sense when made together, every single one or none (atomicity).



TicketBookingDao.java

```
1. package com.neo.spring.dao;
2.
3. import java.sql.SQLException;
4. import org.springframework.jdbc.core.support.JdbcDaoSupport;
5.
6. public class TicketBookingDao extends JdbcDaoSupport {
7.
8.     public int getAccountId(int userId) throws SQLException {
9.           int accountId = 0;
10.          String query = "SELECT ACCOUNT_ID FROM USER_TABLE WHERE USER_ID="
11.                                                                  + userId;
12.               accountId  = getJdbcTemplate().queryForInt(query);
13.               return accountId;
14.          }
15.
16.     public float getPrice(int ticketId) throws SQLException{
17.          float price=0;
18.          String query="SELECT TICKET_PRICE FROM MOVIE_TICKET WHERE
19.                                                  TICKET_ID="+ticketId;
20.          price = getJdbcTemplate().queryForObject(query, Float.class);
21.               return price;
22.     }
23.
24.     public void deductAmount(int accountId, double amount) throws
25.                                                       SQLException{
26.          String query="UPDATE ACCOUNT SET BAL=BAL-"+amount+" WHERE
27.                                                  ACCNO="+accountId;
28.               getJdbcTemplate().execute(query);
29.          }
30.
31.     public void reduceTicketCount(int ticketId, int noOfTickets) throws
32.                                                       SQLException{
33.          String query="UPDATE MOVIE_TICKET SET TICKET_COUNT= TICKET_COUNT-
34.                              "+noOfTickets+" WHERE TICKET_ID="+ticketId;
35.               getJdbcTemplate().execute(query);
```

```
36.         }
37.   }
```

TicketBookingService.java

```
1. package com.neo.spring.service;
2.
3. import org.springframework.transaction.annotation.Propagation;
4. import org.springframework.transaction.annotation.Transactional;
5. import com.neo.spring.dao.TicketBookingDao;
6. @Transactional(propagation=Propagation.REQUIRED)
7. public class TicketBookingService {
8.    private TicketBookingDao dao;
9.
10.         public void setDao(TicketBookingDao dao) {
11.               this.dao = dao;
12.         }
13.
14.         public void bookTicket(int userId, int ticketId, int
15.                                                      totalNoTickets) {
16.               try {
17.                     int accountId = dao.getAccountId(userId);
18.                     float price = dao.getPrice(ticketId);
19.                     double totalAmount = price * totalNoTickets;
20.                     dao.deductAmount(accountId, totalAmount);
21.                     dao.reduceTicketCount(ticketId, totalNoTickets);
22.               } catch (Exception e) {
23.                     System.out.println("...." + e.getMessage());
24.               }
25.         }
26.   }
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:p="http://www.springframework.org/schema/p"
5.    xmlns:tx="http://www.springframework.org/schema/tx"
6.    xsi:schemaLocation="http://www.springframework.org/schema/beans
7.
8.     http://www.springframework.org/schema/beans/spring-beans.xsd
9.     http://www.springframework.org/schema/tx
10.     http://www.springframework.org/schema/tx/spring-tx.xsd">
11.
12.         <tx:annotation-driven transaction-manager="tm" />
13.         <bean id="tm"
14.         class="org.springframework.jdbc.datasource.DataSourceTransaction
15.                                                      Manager">
16.               <property name="dataSource" ref="ds" />
17.         </bean>
18.
19.         <bean id="ds"
20.   class="org.springframework.jdbc.datasource.DriverManagerDataSource">
21.               <property name="driverClassName">
22.                     <value>oracle.jdbc.driver.OracleDriver</value>
23.               </property>
```

```
24.               <property name="url">
25.                     <value>jdbc:oracle:thin:@localhost:1521:server
26.                     </value>
27.               </property>
28.               <property name="username">
29.                     <value>scott</value>
30.               </property>
31.               <property name="password">
32.                     <value>tiger</value>
33.               </property>
34.         </bean>
35.         <bean id="bookingDaoRef"
36.                               class="com.neo.spring.dao.TicketBookingDao">
37.               <property name="dataSource" ref="ds" />
38.         </bean>
39.
40.         <bean id="bookingServiceRef"
41.                         class="com.neo.spring.service.TicketBookingService">
42.               <property name="dao" ref="bookingDaoRef" />
43.         </bean>
44.
45.   </beans>
```

Client.java

```
1. package com.neo.spring.client;
2. import org.springframework.context.ApplicationContext;
3. import org.springframework.context.support.ClassPathXmlApplicationContext;
4. import com.neo.spring.service.TicketBookingService;
5.
6. public class Client {
7.        private static ApplicationContext context = new
8.                                              ClassPathXmlApplicationContext(
9.               "com/neo/spring/config/applicationContext.xml");
10.
11.         public static void main(String[] args) {
12.
13.               TicketBookingService service =
14.               (TicketBookingService)context.getBean("bookingServiceRef");
15.               service.bookTicket(9001, 5001, 5);
16.               System.out.println("Successfully tickets are booked ");
17.         }
 18.  }
```

OUTPUT

```
Successfully tickets are booked
```

ACCOUNT

ACCNO	NAME	BAL
1001	SEKHAR	2325
1002	SOMU	4250

USER_TABLE

USER_ID	ACCOUNT_ID
9001	1001
9002	1002

MOVIE_TICKET

TICKET_ID	TICKET_PRICE	TICKET_COUNT
5001	45	285
5002	30	125

Transaction Attributes in Spring

In spring, Transaction attribute defines the rules for applying policies on methods. It can have one or more parameters from the following:

1. Propagation behavior
2. Isolation level
3. Read only hints
4. Transaction timeout period

Propagation behavior is used to define the transaction boundaries.. It's very much inherited from EJBs transaction management. ADF Bounded task flows do have similar stuff. Spring has a much richer se of propagation behavior managment compare to EJBs. In brief these are:

a)PROPAGATION_MANDATORY: signifies method must be running in a transaction
b)PROPAGATION_NESTED: this is not in EJBs. It talks about creating a new child transaction if a transaction is already existing.
c)PROPAGATION_NEVER: signifies that method should not be running in a transactional context.
d)PROPAGATION_NOT_SUPPORTED: same as above but above will throw exception in case it happens. This will suspend the transaction for the duration the method runs.
e)PROPAGATION_REQUIRED: there must be a transactional context present if not there it will create a new.
f)PROPAGATION_REQUIRES_NEW: there should not be any pre-existing transactional context. it will create a new.
g)PROPAGATION_SUPPORTS: it signifies that it does not need a transactional context but if there is any it doesnt mind.

Isolation level: This has something to do with the concurrency control. When multiple transactions are running they may cause dirty reads, non repeatable reads and phantom reads. Spring provides various options depending upon the tolerance of the business use case you can use one. these are:

a)ISOLATION_DEFAULT: it uses the undelying datastore isolation policy.

b) ISOLATION_READ_UNCOMMITTED: it allows to read the data even when there are uncommited rows thus could lead dirty read or non repeateable read or phantom reads.

c) ISOLATION_READ_COMMITTED: only commited rows are read but still it may cause non-repeateable read or phantom reads.

d) ISOLATION_REPEATABLE_READ: as name suggests prevents dirty read as well as non-repeateable issues but phantom reads could be there.

e)ISOLATION_SERIALIZABLE: this brings the ACID (Atomicity, concurrancy, isolation and durability) compliance. But it is very strict and restricted so could be a performace issue.

Read only hints: Sometimes there is a requirement to just read the data from the datastore. in that case providing this attribute can make a good performance boost as datastore can apply some optimizations. But this can be done by datastore when a new transaction started. so this attribute makes sense only if propagation propety is PROPAGATION_REQUIRED, PROPAGATION_REQUIRES_NEW, or PROPAGATION_NESTED.

Transaction Timeout: This is just a way to prevent deadlocks or unresponsive threads or resource locks in the system. You can specify some timeout that will kill the transaction after the specified timeout period. again it starts when a new transaction begins so it makes sense if propagation is PROPAGATION_REQUIRED, PROPAGATION_REQUIRES_NEW, or PROPAGATION_NESTED.

Spring Web MVC

Q) What is an enterprise application?

- ⇨ Enterprise means business organization
- ⇨ Enterprises (business) provides business services to customer
- ⇨ Any computer application that is used to computerize business Service provided by the enterprise is known as an enterprise application Or simply a business application
  Examples : Banking application, Railway reservation system, Hospital management system, hotel management system ... etc.
- ⇨ Enterprise application can be developed using different languages like java, .net, php ...etc.
- ⇨ In java we can develop enterprise applications using awt, swings, applets, servlets, jsps ...etc.
- ⇨ Enterprise applications are two types
  - o Desktop based applications
  - o Web based applications
- ⇨ Desktop based enterprise applications can be developed using swings, applets ...etc.
- ⇨ Desktop based application services can't be accessible over the web. So we can't get business services 24X7. To get business services 24X7 we have to use web based applications.
- ⇨ Web based Enterprise applications can be developed using **servlets, jsps**.
- ⇨ Now we have to learn which is the best way to develop web based enterprise application.

Enterprise Application developing with Servlet technology

Input.html => CalculateInterestServlet.java

Input.jsp

- ⇨ Enter a/c no and submit the form.

InterestCalculateServlet.java

- ⇨ Receives user input
- ⇨ Validate user input (if any improper data, redisplay input.html with validation errors)
- ⇨ Get connection
- ⇨ Execute query
- ⇨ Get balance
- ⇨ Calculate interest
- ⇨ Display interest

Problems:

- ⇨ Servlet code is not reusable.
- ⇨ Web designing tools can't be used.
- ⇨ Parellel development is not possible

Enterprise Application developed with jsp technology

input.jsp

- ⇨ Enter a/c no t send request to **calucateinterest.jsp**

calucateinterest.jsp

- ⇨ Receives user input
- ⇨ Validate user input (if any improper data redisplay input.jsp with validation errors)
- ⇨ Get connection
- ⇨ Execute query
- ⇨ Get balance
- ⇨ Calculate interest

- ⇨ Display interest

Problems:
- ⇨ Jsp Code not reusable.
- ⇨ Parellel development is not possible.

Enterprise Application developing with jsp, Servlet technologies

Input.jsp => InterestCalculateServlet.java => displayInterest.jsp

Problems
- ⇨ Servlet code is not reusable

To solve the above problems sun microsystem introduced two design patterns for developing enterprise applications
1. Model 1 (Page centric model)
2. Model 2 (MVC)

Q.) What is design pattern?
=> Solution for the recurring problem.
Examples: mvc, dao, singleton, factory, service locator, business delegate, session facade ...etc.
=> **spring mvc** implements MVC design pattern.

Q) what are the different kinds of Logics of web based java application has?
=> Any web based enterprise application has 4 logics.
=> Presentation Logic.
=> Application Logic.
=> Business Logic.
=> Persistence logic. (Data access logic).

1) Presentation Logic:
A real time web applications jsp contains the following things.
- ⇨ HTML Tags.
- ⇨ JSP Tags
  - o EL
  - o Custom Tags
  - o JSTL Tags
- ⇨ Web application Framework given Tags.

Note: JSP, Velocity, Freemarker...etc technologies are used to implement presentation logic.

2) Application Logic:
Code that performs the followings tasks is known as application logic.
1) Receiving the client Request
2) Capturing user input.
3) Performing server side validation on user input.
4) Controlling the flow of application.

Note: Servlet technology is best suited for application logic.

3) Business Logic:
Programatic implemetaion of business rules is nothing but Business Logic.

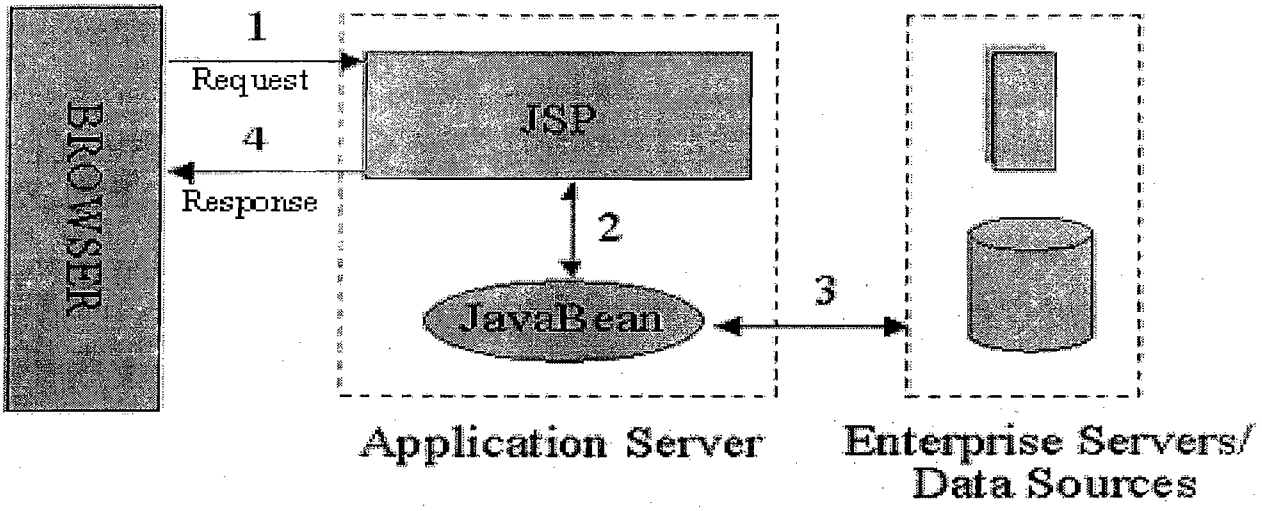
Note: Currently in the industry **java bean, Spring bean & Enterprise Java Bean(EJB)** used for implementing Business Logic.

4.) persistence logic

The code which is used to deal(read, write, update, delete ...etc.) with database is known as Persistence logic or Data Access Logic.

Note: JDBC, HIBERNATE, JPA ...etc are used to implement persistence logic.

Model 1 or page centric model

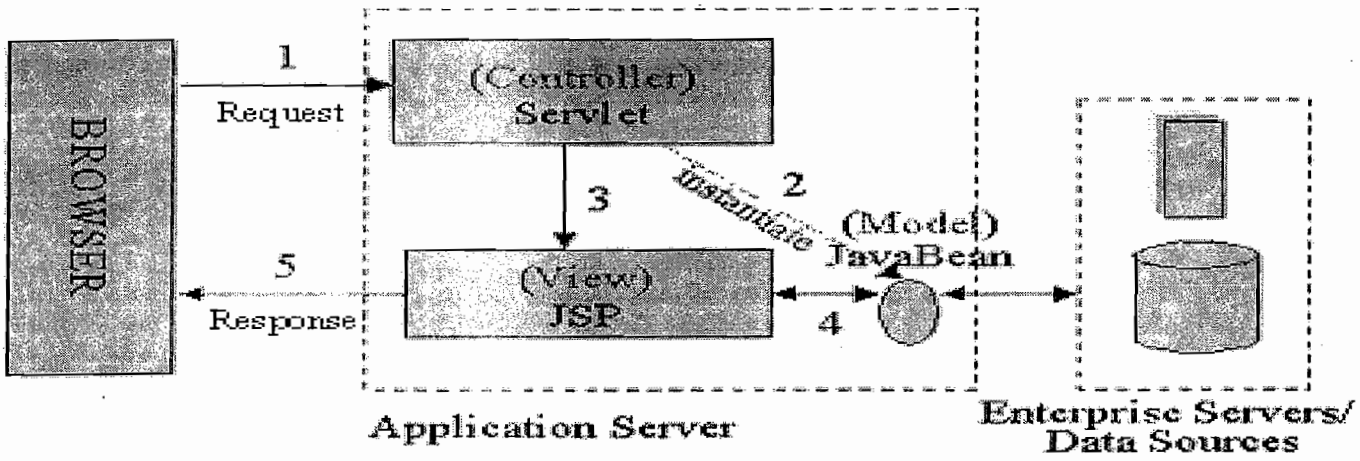


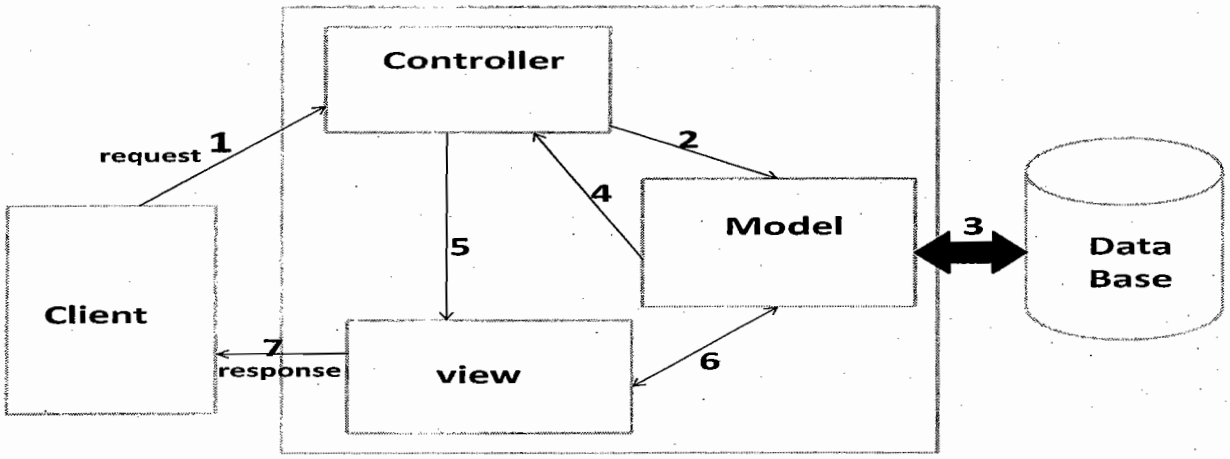
1. Request received by jsp sent by the browser. And capture the user input.
2. Create java bean object which contains business logic. And call the business method.
3. Java bean if required it will communicate with DB to get the required data. Hand over the processed data to the jsp back.
4. Jsp will present the processed data to the end user.

Analysis:

- ⇨ There is no clear separation of responsibilities. Jsp acting as both controller and view. Only business logic is separated using java bean. But application logic and presentation logic is mixed in jsp.
- ⇨ For small scale application development it is advisable to use. But now a days even small scale applications also develeloping with MVC only.

Model 2 or MVC





1- Controller (servlet) receiving the client request, capturing the user i/p and performing server side validations.
2- Controller invoking the business method of the model (java bean).
3- Model contacting the database, getting the business data from the database, processing the data by applying the business rules and holding that processed data.
4- Model giving back the response to Controller.
5- Keep the processed data in heap memory(request/session/servletContext) .
6- Controller switching the control to appropriate view of the application.
7- View (jsp) contacting the heap memory(request/session/servletContext) and getting the processed data.
8- View producing the response (html) screen required for the end user and sending that web page via web server to the web client.

Advantages of MVC

1. Clear separation of responsibilities.
   (Clear separation between presentation logic, application logic and business logic + Data Access Logic.)
2. Code reusability.
3. Single point entry for the application.

4. Easy to test.
5. easy for maintain and future enhancements
6. Parallel development.
7. Supports RAD.
8. Supports multiple view technologies.

Draw backs of MVC

1. Increased complexity to implement controlling logic.
2. Implementing single point of entry (controller) is difficult

⇨ So using MVC implemented framework is advisable. The following are the popular MVC implemented web application frameworks.
- Struts (Apache Software Foundation)
- Spring MVC (spring source)
- JSF(Java Server Faces) (Sun Microsystems)
- ADF(Application Development Framework) (oracle)
- Webwork
- Tapestry
- Wicket ...etc.

Q) What is the purpose of spring mvc?

- spring mvc used to develop the web applications that uses MVC design pattern.
- spring mvc is meant for making web application development faster, cost-effective, and flexible.

Q) what are the java technologies used in a spring application?

- Servlet
- JSP
- Java Bean

Advantages of Spring MVC Framework

⇨ Clear separation of responsibilities. Because it implements MVC design pattern.
⇨ Loose coupling among Model, View and Controller.
⇨ In-built front controller
⇨ Validation implementations is simplified
⇨ Exception logic is simplified
⇨ User input is available in the object oriented format.
⇨ Forward logic is simplified.
⇨ Consistent view support with tiles concept.
⇨ Internationalization(i18n) logic is simplified.
⇨ Spring mvc provides a set of custom JSP tags, which are useful to implement presentation logic.
⇨ Also Spring can integrate effortlessly with other popular Web Frameworks like Struts, WebWork, Java Server Faces and Tapestry.
⇨ Integration with other View technologies like Velocity, Freemarker, Excel orPdf

Q) what is Framework?

A) A Framework is a reusable semi finished application that can be customized to develop a specific application.

Types of frameworks

1. Web frameworks
2. Application frameworks

⇨ Web framework will provide an environment to design and execute only web applications.
- Ex: struts, JSF, Webwork, wicket ...etc.

- ⇨ Application framework will provide an environment to design and execute distributed applications.
  - o Ex: Spring, Jboss seam

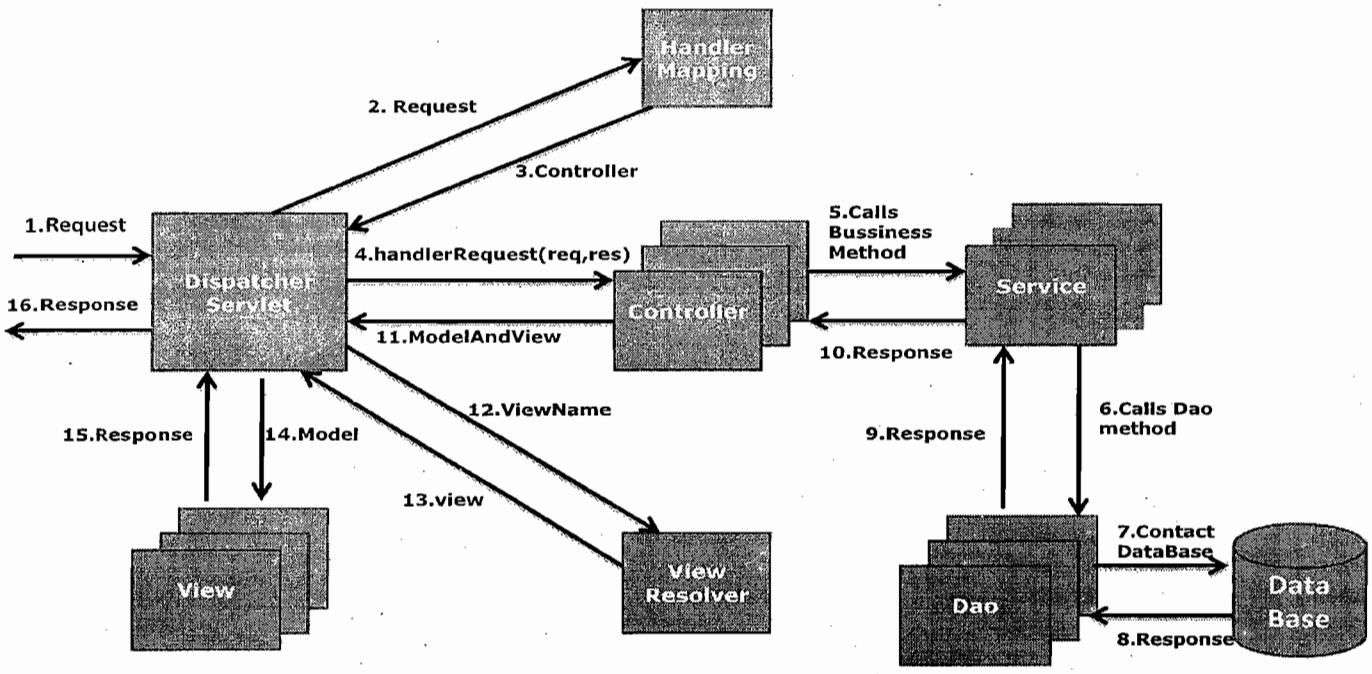
Differences between struts and spring

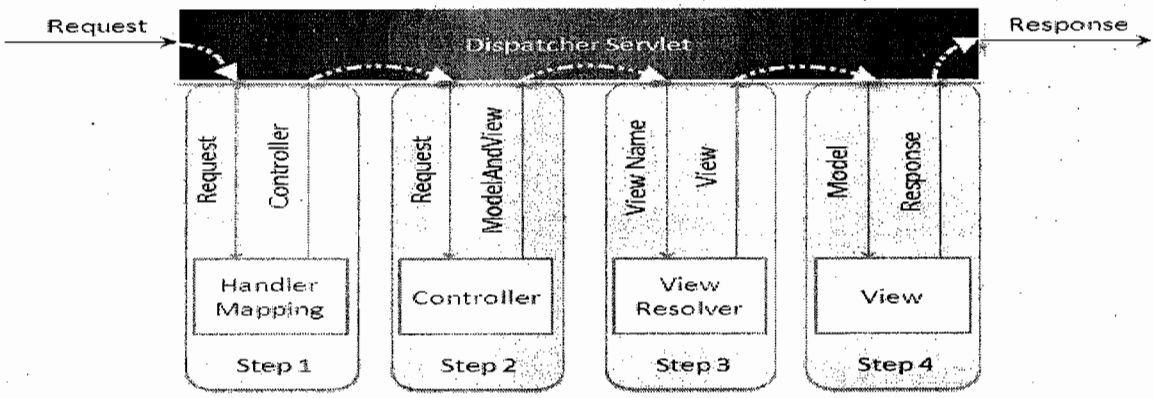
1. Struts is web frame work there by supports only web applications, where as spring is web application framework there by supports both web applications and distributed applications.
2. Struts follows only mvc design pattern. Where as spring implements so many design patterns like singleton, mvc, factory, service locator ...etc.
3. Struts will allow only jsp as view but spring mvc allow all types view technologies like jsp, velocity, freemarker, pdf, excel ...etc.
4. Struts only supports preprocessing, but spring supports both preprocessing and postprocessing.
5. In struts we cannot customize common logic to particular controller. But in spring it supports to customize common logic to particular one controller.
6. Struts will not provide integration built-in support with other technologies, where as spring provides built-in supports integration support with all technologies.
7. Struts is heavy weight framework. Where as spring is light weight framework.
8. Struts is having better tag library support than spring.
9. In struts user data is available in o.o form which is frame work specific object. Where as in spring it provides user data in pojo.

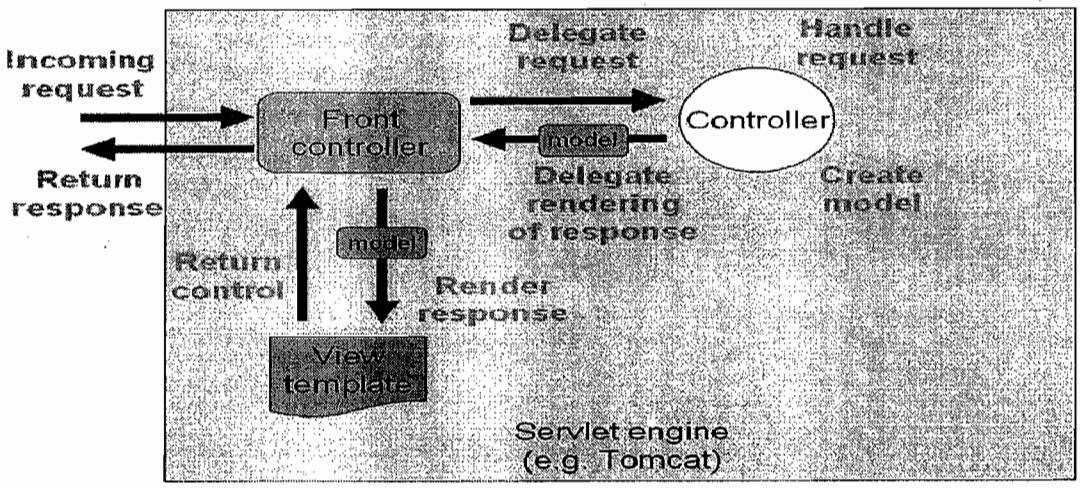
Note :

1. Struts 2.x allow all view technologies.
2. Supports preprocessing and post processing.
3. Provides integration built-in support with other technologies.
4. It is a light weight framework.
5. Struts 1.x has pre-defined support for html, bean, logic, nested, tiles tags ...etc. But in struts 2.x version all the tags are combined as single classification named as 'struts-tags'.

Spring MVC Architecture







Major Components of Spring MVC

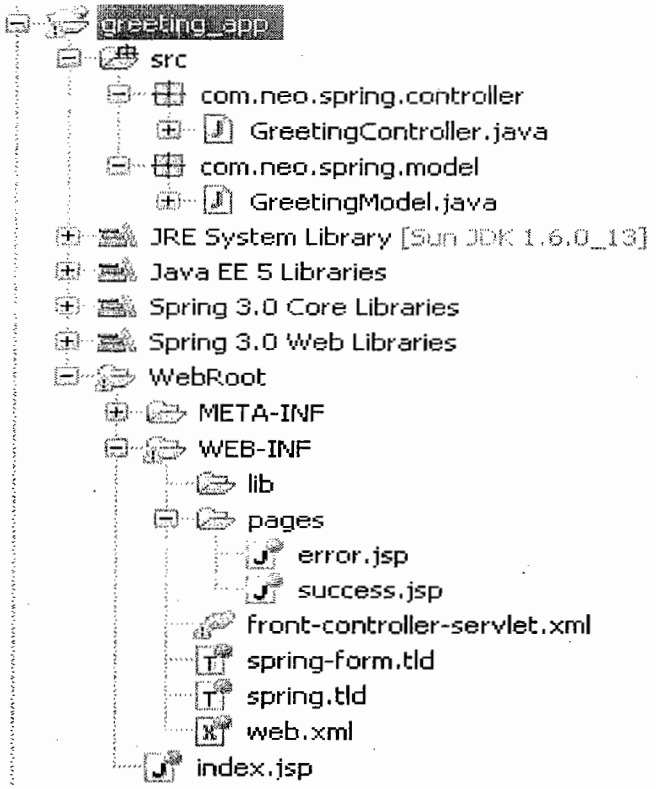
- ***Dispatcher Servlet***
- ***Handler Mappings***
- ***Controller***
- ***ModelAndView***
- ***View Resolver***

The Spring MVC Workflow

1. The Client requests for a ***Resource*** in the Web Application.
2. The **Spring Front Controller** i.e., **DispatcherServlet** made a request to **HandlerMapping** to identify the particular controller for the given url.
3. **HandlerMapping** identifies the controller for the given request and sends to the DispatcherServlet.
4. **DispatcherServlet** will call the **handleRequest(req,res)** method on Controller. Controller is developed by writing a simple java class which implements **Controller** interface or extends its adapter class.
5. **Controller** will call the business method according to bussiness requirement.
6. Service class will calls the Dao class method for the business data.
7. Dao interact with the DataBase to get the database data.
8. Database gives the result.
9. Dao returns the same data to service.
10. Dao given data will be processed according to the business requirements, and returns the results to Controller.
11. The Controller returns the ***Model and the View*** in the form of `ModelAndView` object back to the Front Controller.
12. The ***Front Controller*** i.e., **DispatcherServlet** then tries to resolve the actual ***View*** (which may be Jsp, Velocity or Free marker) by consulting the ***View Resolver object***.
13. ViewResolver selected View is rendered back to the DispatcheServlet.
14. DispatcherServlet consult the particular **View** with the **Model**.
15. View executes and returns HTML output to the DispatcherServlet.
16. DispatcherServlet will sends the output to the Browser.

Note:

- We no need to develop DispatcherServlet just we have to configure in web.xml.
- We just have to configure HandlerMapping and ViewResolver in spring configuration file.
- We have to develop and configure controllers in spring configuration file.



The ***Dispatcher Servlet*** as represented by **org.springframework.web.servlet.DispatcherServlet**, follows the ***Front Controller Design Pattern*** for handling Client Requests. It means that whatever Url comes from the Client, this Servlet will intercept the Client Request before passing the Request Object to the Controller. The ***Web Configurationfile*** i.e., web.xml should be given definition in such a way that this Dispatcher Servlet should be invoked for Client Requests.

Following is the definition given in the web.xml to invoke ***Spring's DispatcherServlet***.

web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app version="2.5"
3.     xmlns="http://java.sun.com/xml/ns/javaee"
4.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5.     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
6.     http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
7.
8.    <servlet>
9.     <servlet-name>front-controller</servlet-name>
10.         <servlet-class>
11.               org.springframework.web.servlet.DispatcherServlet
12.         </servlet-class>
13.     </servlet>
14.
15.     <servlet-mapping>
16.         <servlet-name>front-controller</servlet-name>
17.         <url-pattern>*.sekhar</url-pattern>
18.     </servlet-mapping>
19.
20.     <welcome-file-list>
```

```
21.       <welcome-file>index.jsp</welcome-file>
22.     </welcome-file-list>
23.   </web-app>
```

Look into the definition of `servlet-mapping` tag. It tells that the Client Request (represented by *.sekhar means any Url with an extension .sekhar), invoke the Servlet by name `'front-controller'`. In our case, the `'front-controller'` servlet is nothing but an instance of type `'org.springframework.web.servlet.DispatcherServlet'`.

By default the **DispatcherServlet** will try to look for a file by name `<servlet-name>-servlet.xml` in the `WEB-INF` directory. So, in our case the Servlet will look for a file name called **`front-controller-servlet.xml`** file in the **WEB-INF** directory.

Controller

Controllers are components that are being called by the Dispatcher Servlet for doing any kind of Business Logic. Spring Distribution already comes with a variety of ***Controller Components*** each doing a specific purpose. All Controller Components in Spring implement the `org.springframework.web.servlet.mvc.Controller` interface.
Now here we are using `AbstractController`.

GreetingController.java

```
1. package com.neo.spring.controller;
2.
3. import javax.servlet.http.HttpServletRequest;
4. import javax.servlet.http.HttpServletResponse;
5. import org.springframework.web.servlet.ModelAndView;
6. import org.springframework.web.servlet.mvc.AbstractController;
7. import com.neo.spring.model.GreetingModel;
8.
9. public class GreetingController extends AbstractController {
10.
11.         @Override
12.         protected ModelAndView handleRequestInternal(HttpServletRequest
13.                 request, HttpServletResponse response) throws Exception {
14.               String outputpage = errorView;
15.               String greetingMessage = null;
16.
17.               String name = request.getParameter("name");
18.               if (name == null || name.trim().length() == 0) {
19.                     greetingMessage = model.sayGreetings("Guest");
20.                     outputpage = errorView;
21.               } else {
22.                     greetingMessage = model.sayGreetings(name);
23.                     outputpage = successView;
24.               }
25.
26.               return new ModelAndView(outputpage, "greetingMsg",
27.                                                         greetingMessage);
28.         }
29.
30.         public void setSuccessView(String successView) {
31.               this.successView = successView;
32.         }
33.
```

```
34.         public void setErrorView(String errorView) {
35.               this.errorView = errorView;
36.         }
37.
38.         public void setModel(GreetingModel model) {
39.               this.model = model;
40.         }
41.
42.         private GreetingModel model;
43.         private String successView;
44.         private String errorView;
45.   }
```

Note that the Dispatcher Servlet will call the `handleRequestInternal()` method by passing the Request and the Response parameters. The implementation just returns a `ModelAndView` object with `successView` being the logical view name. There are Components called ***View Resolvers*** whose job is to provide a mapping between the ***Logical View Name*** and the actual ***Physical Location of the View Resource***. For the time being, assume that somehow, `successView` is mapped to `success.jsp`. So, whenever the Dispatcher Servlet invokes this **GreetingController** object, finally `success.jsp` will be rendered back to the Client.

Model And View

Model and View represented by the class `org.springframework.web.servlet.ModelAndView` is returned by the Controller object back to the **DispatcherServlet**. This class is just a Container class for holding the Model and the View information. The Model object represents some piece of information that can be used by the View to display the information. Both these Objects are given high degree of abstraction in the Spring Framework.

Any kind of ***View Technology*** (`org.springframework.web.servlet.View`) can be plugged into the Framework with ease. For example, **Excel, Jasper Reports, Pdf, Xslt, Free Marker, Html, Tiles, Velocity etc**. are the supported Frameworks as of now. The Model object (represented by `org.springframework.ui.ModelMap`) is internally maintained as a Map for storing the Information.

Ex:-

```
ModelAndView mv=new ModelAndView("successView", "greetingMsg",
                                                    greetingMessage);
```

Note, in the above example, a string with "`successView`" is passed for the View. This way of specifying a ***View*** is called a ***Logical View***. It means that `successView` either can point to something called `success.jsp` or `success.pdf` or `success.xml`. The ***Physical View Location*** corresponding to the ***Logical View*** can be made configurable in the Configuration File.

GreetingModel.java

```
1. package com.neo.spring.model;
2.
3. public class GreetingModel {
4.    public String sayGreetings(String name) {
5.          return "Welcome : " + name;
6.    }
7. }
```

index.jsp

```
1. <html>
```

```
2.   <body bgcolor="pink" >
3.     <center>
4.          <h1>Spring MVC Greeting Application</h1>
5.          <hr>
6.          <form action="./greet.sekhar" method="post" >
7.               Name : <input type="text" name="name"  > <br/>
8.               <input type="submit" value="Get Greetings" >
9.          </form>
10.       </center>
11.     </body>
12.   </html>
```

success.jsp

```
1. <html>
2.   <body bgcolor="pink" >
3.     <center>
4.          <h1>Spring MVC Greeting Application</h1>
5.          <hr>
6.          Message : <%= request.getAttribute("greetingMsg") %> <br/>
7.          Message : ${greetingMsg}
8.     </center>
9.   </body>
10.   </html>
```

error.jsp

```
1. <html>
2.   <body bgcolor="pink" >
3.     <center>
4.          <h1>Spring MVC Greeting Application</h1>
5.          <hr>
6.          Message : <%= request.getAttribute("greetingMsg") %> <br/>
7.          Message : ${greetingMsg}
8.     </center>
9.   </body>
10.   </html>
```

front-controller-servlet.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xmlns:p="http://www.springframework.org/schema/p"
5.     xsi:schemaLocation="http://www.springframework.org/schema/beans
6.     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7.
8.     <bean id="gm" class="com.neo.spring.model.GreetingModel"></bean>
9.
10.         <bean name="/greet.sekhar"
11.               class="com.neo.spring.controller.GreetingController">
12.               <property name="successView" value="success"></property>
13.               <property name="errorView" value="error"></property>
14.               <property name="model" ref="gm"></property>
15.         </bean>
16.
17.         <bean id="bhm"
18.         class="org.springframework.web.servlet.handler.BeanNameUrlHandler
```

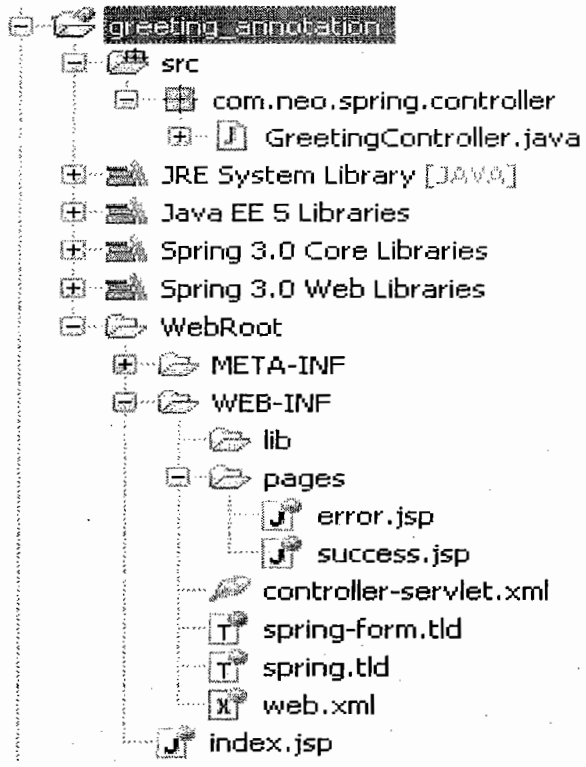
```
19.         Mapping"></bean>
20.
21.         <bean id="irvr"
22.         class="org.springframework.web.servlet.view.InternalResourceView
23.         Resolver">
24.               <property name="prefix" value="/WEB-INF/pages/"></property>
25.               <property name="suffix" value=".jsp"></property>
26.         </bean>
27.   </beans>
```

The ***Internal Resource View Resolver*** will try to map the Logical name of the Resource as returned by the Controller object in the form of ModelAndView object to the ***Physical View location***. In our **GreetingController** class definition which returns different ModelAndView objects.

Assume that in our example, if the Client Request satisfies if() condition then the view `error.jsp` which is present in the `/WEB-INF/pages` folder should be displayed and for the client Requests not satisfying if() condition then `success.jsp` should be displayed.

BeanNameUrlHandlerMapping is the simplest of the *Handler Mapping* and it is used to map the Url that comes from the Clients directly to the Bean Object.

Annotation Spring MVC



web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app version="2.5"
3.     xmlns="http://java.sun.com/xml/ns/javaee"
4.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5.     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
6.     http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
7.
8.     <servlet>
9.           <servlet-name>controller</servlet-name>
10.               <servlet-class>
11.                   org.springframework.web.servlet.DispatcherServlet
12.               </servlet-class>
13.          </servlet>
14.
15.          <servlet-mapping>
16.               <servlet-name>controller</servlet-name>
17.               <url-pattern>/</url-pattern>
18.          </servlet-mapping>
19.
20.      <welcome-file-list>
21.        <welcome-file>index.jsp</welcome-file>
22.      </welcome-file-list>
23.   </web-app>
```

In Annotation Spring MVC, The controller class is no longer need to extends base controller like **AbstractController** or **SimpleFormController**, just simply annotate the class with a **@Controller** annotation.
Handler Mapping – No more declaration for the handler mapping like **BeanNameUrlHandlerMapping**, **ControllerClassNameHandlerMapping** or **SimpleUrlHandlerMapping**, all are replaced with a standard **@RequestMapping** annotation.

If the **@RequestMapping** is applied at class level (can apply at method level for multi-actions controller), it required to put a **RequestMethod** to indicate which method to handle the mapping request.
In this case, if an URL "**/greetings**" is requested, it will map to this **GreetingController**, and handle the request with **greet()** method.

GreetingController.java

```
1. package com.neo.spring.controller;
2.
3. import javax.servlet.http.HttpServletRequest;
4. import javax.servlet.http.HttpServletResponse;
5. import org.springframework.stereotype.Controller;
6. import org.springframework.web.bind.annotation.RequestMapping;
7. import org.springframework.web.bind.annotation.RequestMethod;
8. import org.springframework.web.servlet.ModelAndView;
9.
10.   @Controller
11.   public class GreetingController {
12.
13.         @RequestMapping(value="/test",method=RequestMethod.GET)
14.         public ModelAndView abc(){
15.               return new ModelAndView("success",
16.                                                "greetingMsg","Testing");
17.         }
18.
19.         @RequestMapping(value = "/greetings",method=RequestMethod.POST)
20.         public ModelAndView greet(HttpServletRequest request,
21.                     HttpServletResponse response) {
22.               String outputView = "error";
23.               String name = request.getParameter("name");
24.               String greetingMessage = null;
25.               if (name == null || name.trim().length() == 0) {
26.                     outputView = "error";
27.                     greetingMessage = "Welcome Guest";
28.
29.               }else{
30.                     outputView= "success";
31.                     greetingMessage="Welcome "+name;
32.               }
33.
34.               return new ModelAndView(outputView, "greetingMsg",
35.                                                      greetingMessage);
36.         }
37.   }
```

index.jsp

```
1. <html>
```

```
2.   <body bgcolor="pink" >
3.     <center>
4.          <h1>Spring MVC Greeting Application</h1>
5.          <hr>
6.          <form action="./greetings" method="post" >
7.               Name : <input type="text" name="name"  > <br/>
8.               <input type="submit" value="Get Greetings" >
9.          </form>
10.   <br>
11.   <h2> <a href="./test">Call abc() method </a> </h2>
12.       </center>
13.     </body>
14.   </html>
```

success.jsp

```
1. <html>
2.   <body bgcolor="pink" >
3.     <center>
4.          <h1>Spring MVC Greeting Application</h1>
5.          <hr>
6.          <h2>Success Page</h2>
7.          Message : <%= request.getAttribute("greetingMsg") %> <br/>
8.          Message : ${greetingMsg}
9.     </center>
10.     </body>
11.   </html>
```

error.jsp

```
1. <html>
2.   <body bgcolor="pink" >
3.     <center>
4.          <h1>Spring MVC Greeting Application</h1>
5.          <hr>
6.          <h2>Error Page</h2>
7.          Message : <%= request.getAttribute("greetingMsg") %> <br/>
8.          Message : ${greetingMsg}
9.     </center>
10.     </body>
11.   </html>
```

To work with annotation, we have to enable the **Spring's auto component scanning** feature through the **<context:component-scan>** element and we have to provide the location of controllers in **base-package** attribute.
And still we have to configure **ViewResolver.**

controller-servlet.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xmlns:context="http://www.springframework.org/schema/context"
5.     xsi:schemaLocation="http://www.springframework.org/schema/beans
6.     http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
7.     http://www.springframework.org/schema/context
8.     http://www.springframework.org/schema/context/spring-context-2.5.xsd">
```

```
9.
10.   <context:component-scan base-package="com.neo.spring.controller" />
11.         <bean id="viewResolver"
12.         class="org.springframework.web.servlet.view.InternalResource
13.                                                         ViewResolver">
14.               <property name="prefix">
15.                     <value>/WEB-INF/pages/</value>
16.               </property>
17.               <property name="suffix">
18.                     <value>.jsp</value>
19.               </property>
20.         </bean>
21.   </beans>
```

HandlerMapping

BeanNameUrlHandlerMapping:

BeanNameUrlHandlerMapping is the default handler mapping class, which maps the **URL requests to the names of the beans**. Additionally, the mapping is support the Ant style regex pattern match as well. For example,

```
<beans ...>
   <bean      class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>

   <bean name="/welcome.htm"
            class="com.neo.spring.controller.WelcomeController" />

   <bean name="/streetName.htm"
            class="com.neo.spring.controller.StreetNameController" />

   <bean name="/process*.htm"
            class="com.neo.spring.controller.ProcessController" />
</beans>
```

In above example, If URL pattern

1. **/welcome.htm** is requested, DispatcherServlet will forward the request to the "**WelcomeController**".
2. **/streetName.htm** is requested, DispatcherServlet will forward the request to the "**StreetNameController**".
3. **/processCreditCard.htm** or **/process{any thing}.htm** is requested, DispatcherServlet will forward the request to the "**ProcessController**".

Note

Actually, we don't need to define the **BeanNameUrlHandlerMapping** in the spring configuration, by default, if no handler mapping can be found, the DispatcherServlet will creates a **BeanNameUrlHandlerMapping** automatically. So, the above spring configuration file is equivalence to the following configuration file:

```
<beans ...>
    <bean name="/welcome.htm"
            class="com.neo.spring.controller.WelcomeController" />

   <bean name="/streetName.htm"
            class="com.neo.spring.controller.StreetNameController" />

   <bean name="/process*.htm"
            class="com.neo.spring.controller.ProcessController" />
 </beans>
```

ControllerClassNameHandlerMapping:

ControllerClassNameHandlerMapping is a handler mapping class, which use convention to maps the requested URL to Controller (convention over configuration). It takes the Class name, remove the 'Controller' suffix if it exists and return the remaining text, lower-cased and with a leading "/".

By default, Spring MVC is using the **BeanNameUrlHandlerMapping** handler mapping.

```
<beans ...>
```

```
  <bean name="/welcome.htm"
          class="com.neo.spring.controller.WelcomeController" />

  <bean name="/helloGuest.htm"
          class="com.neo.spring.controller.HelloGuestController" />
 </beans>
```

To enable the **ControllerClassNameHandlerMapping**, declared it in the bean configuration file, and **the controller's bean's name is no longer required**.

```
<beans ...>

  <bean
class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping" />

  <bean class="com.neo.spring.controller.WelcomeController" />

  <bean class="com.neo.spring.controller.HelloGuestController" />
</beans>
```

Now, the application is mapping the requested URL by the following conventions.

```
WelcomeController     -> /welcome*
HelloGuestController -> /helloguest*
```

1. /welcome.htm –> WelcomeController.
2. /welcomeHome.htm –> WelcomeController.
3. /helloguest.htm –> HelloGuestController.
4. /helloguest12345.htm –> HelloGuestController.
5. /helloGuest.htm, failed to map **/helloguest***, the "g" case is not match.

2. Case sensitive

To solve the case sensitive issue stated above, declared the "**caseSensitive**" property and set it to true.

```
<beans ...>

  <bean
class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping" >
       <property name="caseSensitive" value="true" />
  </bean>

  <bean class="com.neo.spring.controller.WelcomeController" />

  <bean class="com.neo.spring.controller.HelloGuestController" />

</beans>
```

Now, the application is mapping the requested URL by the following conventions.

```
WelcomeController     -> /welcome*
HelloGuestController -> /helloGuest*
```

1. /helloGuest.htm –> HelloGuestController.
2. /helloguest.htm, failed to map "/helloGuest*", the "G" case is not match.

3. pathPrefix

Additionally, you can specify a prefix to maps the requested URL, declared a "**pathPrefix**" property.

```
<beans ...>

  <bean
class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping" >
        <property name="caseSensitive" value="true" />
        <property name="pathPrefix" value="/customer" />
  </bean>

  <bean class="com.neo.spring.controller.WelcomeController" />

  <bean class="com.neo.spring.controller.HelloGuestController" />

</beans>
```

Now, the application is mapping the requested URL by the following conventions.

```
WelcomeController     -> /customer/welcome*
HelloGuestController -> /customer/helloGuest*
```

1. /customer/welcome.htm –> WelcomeController.
2. /customer/helloGuest.htm –> HelloGuestController.
3. /welcome.htm, failed.
4. /helloGuest.htm, failed.

SimpleUrlHandlerMapping:

In Spring MVC application, the **SimpleUrlHandlerMapping** is the most flexible handler mapping class, which allows developer to specify the mapping of URL pattern and handlers explicitly.
The **SimpleUrlHandlerMapping** can be declared in two ways.

1. prop key

The property keys are the URL patterns while the property values are the handler IDs or names.

```
<beans ...>

        <bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
           <property name="mappings">
                <props>
                   <prop key="/welcome.htm">welcomeController</prop>
                   <prop key="/*/welcome.htm">welcomeController</prop>
                   <prop key="/helloGuest.htm">helloGuestController</prop>
                 </props>
           </property>
        </bean>

        <bean id="welcomeController"
                class="com.neo.spring.controller.WelcomeController" />

        <bean id="helloGuestController"
                class="com.neo.spring.controller.HelloGuestController" />
 </beans>
```

2. value

The left side are the URL patterns while the right side are the handler IDs or names, separate by a equal symbol "=".

```
<beans ...>
```

```
        <bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
           <property name="mappings">
                <value>
                   /welcome.htm=welcomeController
                   /*/welcome.htm=welcomeController
                   /helloGuest.htm=helloGuestController
                </value>
           </property>
        </bean>

        <bean id="welcomeController"
                class="com.neo.spring.controller.WelcomeController" />

        <bean id="helloGuestController"
                class="com.neo.spring.controller.HelloGuestController" />

</beans>
```

Both are defined the same handler mappings.

1. /welcome.htm -> welcomeController.
2. /{anything}/welcome.htm -> welcomeController.
3. /helloGuest.htm -> helloGuestController.

Configure the handler mapping priority in Spring MVC

Often times, you may mix use of multiple handler mappings strategy in Spring MVC development. For example, use **ControllerClassNameHandlerMapping** to map all the convention handler mappings, and **SimpleUrlHandlerMapping** to map other special handler mappings explicitly.
For example,

```
<beans ...>

   <bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
      <property name="mappings">
        <value>
          /index.htm=welcomeController
          /welcome.htm=welcomeController
          /main.htm=welcomeController
          /home.htm=welcomeController
        </value>
      </property>
      <property name="order" value="0" />
   </bean>

   <bean
class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping" >
      <property name="caseSensitive" value="true" />
      <property name="order" value="1" />
   </bean>

   <bean id="welcomeController"
      class="com.neo.spring.controller.WelcomeController" />

   <bean class="com.neo.spring.controller.HelloGuestController" />

</beans>
```

In above case, it's important to specify the handler mapping priority, so that it won't cause the conflict. We can set the priority via the "**order**" property, where the lower order value has the higher priority.

Spring MVC Handler Interceptors

Spring MVC allow you to intercept web request through handler interceptors. The handler interceptor have to implement the **HandlerInterceptor** interface, which contains three methods :

1. **preHandle()** – Called before the handler execution, returns a boolean value,
   "true" : continue the handler execution chain;
   "false" : stop the execution chain and return it.

2. **postHandle()** – Called after the handler execution, allow manipulate the ModelAndView object before render it to view page.

3. **afterCompletion()** – Called after the complete request has finished. Seldom use, cant find any use case.

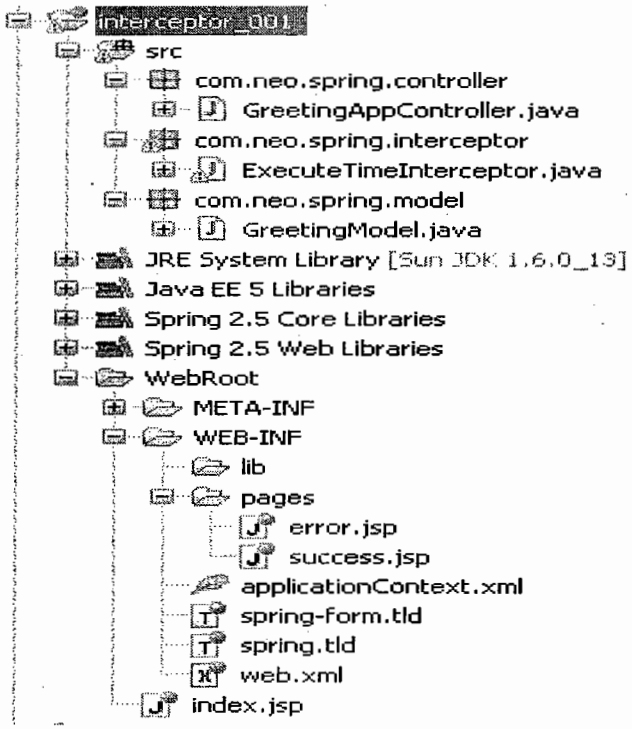
In our application, we are creating two handler interceptors to show the use of the **HandlerInterceptor**.

1. **ExecuteTimeInterceptor** – Intercept the web request, and log the controller execution time.
2. **MaintenanceInterceptor** – Intercept the web request, check if the current time is in between the maintenance time, if yes then redirect it to maintenance page.

Note
It's recommended to extend the **HandlerInterceptorAdapter** for the convenient default implementations.

ExecuteTimeInterceptor
Intercept the before and after controller execution, log the start and end of the execution time, save it into the existing intercepted controller's modelAndView for later display.

To enable Handler Interceptor, configure our handler interceptor class in the HandlerMapping "**interceptors**" property.



ExecuteTimeInterceptor.java

```
1. package com.neo.spring.interceptor;
2.
3. import java.util.Map;
4. import javax.servlet.http.HttpServletRequest;
5. import javax.servlet.http.HttpServletResponse;
6. import org.springframework.web.servlet.ModelAndView;
7. import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;
8.
9. public class ExecuteTimeInterceptor extends HandlerInterceptorAdapter {
10.
11.         @Override
12.         public boolean preHandle(HttpServletRequest request,
13.                     HttpServletResponse response, Object handler) throws
14.                                                                 Exception {
15.               System.out.println("ExecuteTimeInterceptor.preHandle()");
16.               long startTime = System.currentTimeMillis();
17.               request.setAttribute("startTime", startTime);
18.               return true;
19.         }
20.
21.         @Override
22.         public void postHandle(HttpServletRequest request,
23.               HttpServletResponse response, Object handler,
24.               ModelAndView modelAndView) throws Exception {
25.               System.out.println("------------
26.                           ExecuteTimeInterceptor.postHandle()--------");
27.               long endTime = System.currentTimeMillis();
28.               long startTime = (Long) request.getAttribute("startTime");
29.               System.out.println("Time taken to exeucte " +
30.                        handler.getClass()+ " : " + (endTime - startTime));
31.               System.out.println("View Name : " +
32.                                             modelAndView.getViewName());
33.               Map<String, Object> map = modelAndView.getModel();
34.               System.out.println("Model : " + map);
35.               System.out.println("------------
36.                           ExecuteTimeInterceptor.postHandle()--------");
37.         }
38.
39.         @Override
40.         public void afterCompletion(HttpServletRequest request,
41.                     HttpServletResponse response, Object handler,
42.                                       Exception ex)throws Exception {
43.
44.   System.out.println("ExecuteTimeInterceptor.afterCompletion()");
45.         }
46.   }
```

web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5.     http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
```

```
6.    <servlet>
7.          <servlet-name>front-controller</servlet-name>
8.          <servlet-class>
9.                org.springframework.web.servlet.DispatcherServlet
10.         </servlet-class>
11.         <init-param>
12.               <param-name>contextConfigLocation</param-name>
13.               <param-value>/WEB-INF/applicationContext.xml
14.               </param-value>
15.         </init-param>
16.         </servlet>
17.
18.         <servlet-mapping>
19.               <servlet-name>front-controller</servlet-name>
20.               <url-pattern>/</url-pattern>
21.         </servlet-mapping>
22.
23.         <listener>
24.               <listener-class>
25.                     org.springframework.web.context.ContextLoaderListener
26.               </listener-class>
27.         </listener>
28.
29.         <welcome-file-list>
30.               <welcome-file>index.jsp</welcome-file>
31.         </welcome-file-list>
32.   </web-app>
```

GreetingAppController.java

```
1. package com.neo.spring.controller;
2. import javax.servlet.http.HttpServletRequest;
3. import javax.servlet.http.HttpServletResponse;
4. import org.springframework.web.servlet.ModelAndView;
5. import org.springframework.web.servlet.mvc.AbstractController;
6. import com.neo.spring.model.GreetingModel;
7.
8. public class GreetingAppController extends AbstractController {
9.
10.         @Override
11.         protected ModelAndView handleRequestInternal(HttpServletRequest
12.                 request, HttpServletResponse response) throws Exception {
13.
14.      System.out.println("GreetingAppController.handleRequestInternal()");
15.               String outputpage = errorView;
16.               String greetingMessage = null;
17.
18.               String name = request.getParameter("name");
19.               if (name == null || name.trim().length() == 0) {
20.                     greetingMessage = model.sayGreetings("Guest");
21.                     outputpage = errorView;
22.               } else {
23.                     greetingMessage = model.sayGreetings(name);
24.                     outputpage = succssView;
25.               }
26.
```

```
27.                 return new ModelAndView(outputpage, "greetingMsg",
28.                                                       greetingMessage);
29.           }
30.
31.           public void setSuccssView(String succssView) {
32.                 this.succssView = succssView;
33.           }
34.
35.           public void setErrorView(String errorView) {
36.                 this.errorView = errorView;
37.           }
38.
39.           public void setModel(GreetingModel model) {
40.                 this.model = model;
41.           }
42.
43.           private GreetingModel model;
44.           private String succssView;
45.           private String errorView;
46.
47.     }
```

GreetingModel.java

```
1. package com.neo.spring.model;
2.    public class GreetingModel {
3.     public String sayGreetings(String name) {
4.           return "Welcome : " + name;
5.     }
6. }
```

index.jsp

```
1. <html>
2.    <body bgcolor="pink" >
3.      <center>
4.           <h1>Spring MVC Greeting Application</h1>
5.           <hr>
6.           <form action="./greetings" method="post" >
7.                 Name : <input type="text" name="name"  > <br/>
8.                 <input type="submit" value="Get Greetings" >
9.           </form>
10.        </center>
11.     </body>
12.   </html>
```

success.jsp

```
1. <html>
2.    <body bgcolor="pink" >
3.      <center>
4.           <h1>Spring MVC Greeting Application</h1>
5.           <hr>
6.           <h2>Sucess Page</h2>
7.           Message : <%= request.getAttribute("greetingMsg") %> <br/>
8.           Message : ${greetingMsg}
9.     </center>
10.     </body>
```

```
11.   </html>
```

error.jsp

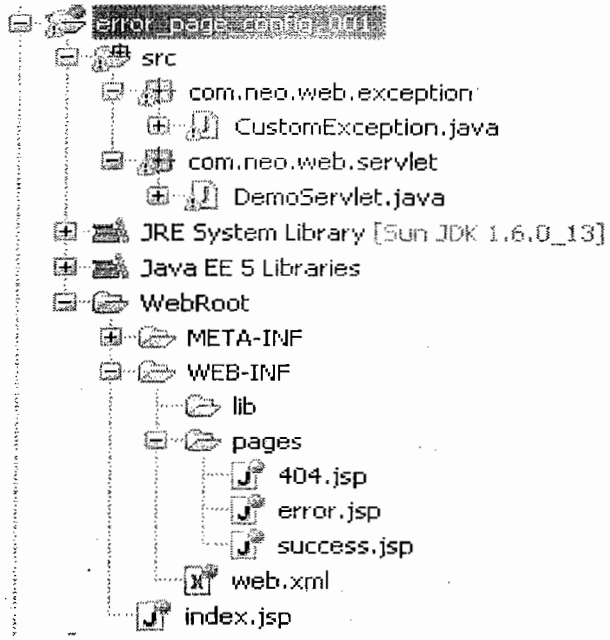
```
1. <html>
2.   <body bgcolor="pink" >
3.     <center>
4.          <h1>Spring MVC Greeting Application</h1>
5.          <hr>
6.          <h2>Error Page</h2>
7.          Message : <%= request.getAttribute("greetingMsg") %> <br/>
8.          Message : ${greetingMsg}
9.     </center>
10.     </body>
11.   <html>
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xmlns:p="http://www.springframework.org/schema/p"
5.     xsi:schemaLocation="http://www.springframework.org/schema/beans
6.     http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
7.
8. <bean id="gm" class="com.neo.spring.model.GreetingModel" />
9.
10.    <bean id="gc" class="com.neo.spring.controller.GreetingAppController">
11.          <property name="succssView" value="success" />
12.          <property name="errorView" value="error" />
13.          <property name="model" ref="gm" />
14.    </bean>
15.
16.    <bean id="eti"
17.    class="com.neo.spring.interceptor.ExecuteTimeInterceptor" />
18.
19.    <bean id="suhm" class="org.springframework.web.servlet.handler.Simple
20.                                                    UrlHandlerMapping">
21.          <property name="mappings">
22.                <props>
23.                      <prop key="/greetings">gc</prop>
24.                </props>
25.          </property>
26.          <property name="interceptors">
27.                <list>
28.                      <ref bean="eti" />
29.                </list>
30.          </property>
31.    </bean>
32.
33.    <bean id="vr"
34.    class="org.springframework.web.servlet.view.InternalResource
35.                                                    ViewResolver">
36.          <property name="prefix" value="/WEB-INF/pages/" />
37.          <property name="suffix" value=".jsp" />
38.    </bean>
39.  </beans>
```

Configuring Exceptions in web.xml

In servlet web application, it's always recommended to display a custom friendly error page instead of the default long java plain exception code.
In web.xml, we can configure a custom error page to map a specified error code or exception:



DemoServlet.java

```
1. package com.neo.web.servlet;
2. import java.io.IOException;
3. import javax.servlet.ServletException;
4. import javax.servlet.http.HttpServlet;
5. import javax.servlet.http.HttpServletRequest;
6. import javax.servlet.http.HttpServletResponse;
7. import com.neo.web.exception.CustomException;
8.
9. public class DemoServlet extends HttpServlet {
10.         public void doGet(HttpServletRequest request, HttpServletResponse
11.                           response)throws ServletException, IOException {
12.               String type = request.getParameter("type");
13.               if ("CustomException".equalsIgnoreCase(type)) {
14.                     throw new CustomException("user defined exception");
15.               } else if ("AIOB".equalsIgnoreCase(type)) {
16.                     throw new ArrayIndexOutOfBoundsException(
17.                     "testing error page for array index out of bounds");
18.               } else if ("success".equalsIgnoreCase(type)) {
19.                      request.getRequestDispatcher("/WEB-INF/pages/
20.                                  success.jsp").forward(request,response);
21.               }
22.         }
23.   }
```

CustomException.java

```
1. package com.neo.web.exception;
2. public class CustomException extends RuntimeException {
3.    public CustomException(String message) {
```

```
4.          super(message);
5.    }
6. }
```

web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
3.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5.    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
6.    <servlet>
7.          <servlet-name>DemoServlet</servlet-name>
8.          <servlet-class>com.neo.web.servlet.DemoServlet</servlet-class>
9.    </servlet>
10.
11.         <servlet-mapping>
12.               <servlet-name>DemoServlet</servlet-name>
13.               <url-pattern>/demo</url-pattern>
14.         </servlet-mapping>
15.         <error-page>
16.               <error-code>404</error-code>
17.               <location>/WEB-INF/pages/404.jsp</location>
18.         </error-page>
19.
20.         <error-page>
21.               <exception-type>
22.                     com.neo.web.exception.CustomException
23.               </exception-type>
24.               <location>/WEB-INF/pages/error.jsp</location>
25.         </error-page>
26.
27.         <error-page>
28.               <exception-type>
29.                     java.lang.ArrayIndexOutOfBoundsException
30.               </exception-type>
31.               <location>/WEB-INF/pages/error.jsp</location>
32.         </error-page>
33.
34.         <welcome-file-list>
35.               <welcome-file>index.jsp</welcome-file>
36.         </welcome-file-list>
37.   </web-app>
```

index.jsp

```
1. <html>
2.   <body bgcolor="pink" >
3. <center>
4.    <h1>Error code demo application</h1>
5.    <hr/>
6.    <form action="./demo">
7.    Type :
8.    <select name="type" >
9.          <option value="CustomException">CustomException</option>
10.         <option value="AIOB">ArrayIndexOutOfBoundsException</option>
11.         <option value="success" >success</option>
```

```
12.   </select> <br/>
13.   <input type="submit" value="Test Me" >
14.   </form>
15.   </center>
16.     </body>
17.   </html>
```

error.jsp

```
1. <%@page isErrorPage="true" %>
2. <html>
3.   <body bgcolor="pink" >
4.     <center>
5.     <h1>Error code demo application</h1>
6.     <hr/>
7.     <h2><%= exception.getMessage() %></h2>
8.     </center>
9.    </body>
10.   </html>
```

404.jsp

```
1. <html>
2.   <body bgcolor="pink" >
3. <center>
4.    <h1>Error code demo application</h1>
5.    <hr/>
6.    <h2>The request url is not correct. Pls try with correct url</h2>
7. </center>
8.   </body>
9. </html>
```

success.jsp

```
1. <%@page isErrorPage="true" %>
2. <html>
3.   <body bgcolor="pink" >
4. <center>
5.    <h1>Error code demo application</h1>
6.    <hr/>
7.    <h2>success</h2>
8. </center>
9.   </body>
10.   </html>
```

Exception Handling in Spring Configuration

If the exception handling function is existed in the servlet container, why we still need to use the Spring to handle the exception?

For two reasons:

1. **Data population** – The servlet container will render the error page directly; While the Spring allow you to populate model or data to the error page, so that you can customize a more user friendly error page.
2. **Business logic** – Spring allow you to apply extra business logic before render the error page, like logging, auditing and etc.

In our example, it shows the use of **SimpleMappingExceptionResolver** to do the exception handling in Spring MVC application.

Declare **SimpleMappingExceptionResolver** in Spring's bean configuration file.
In this case, when

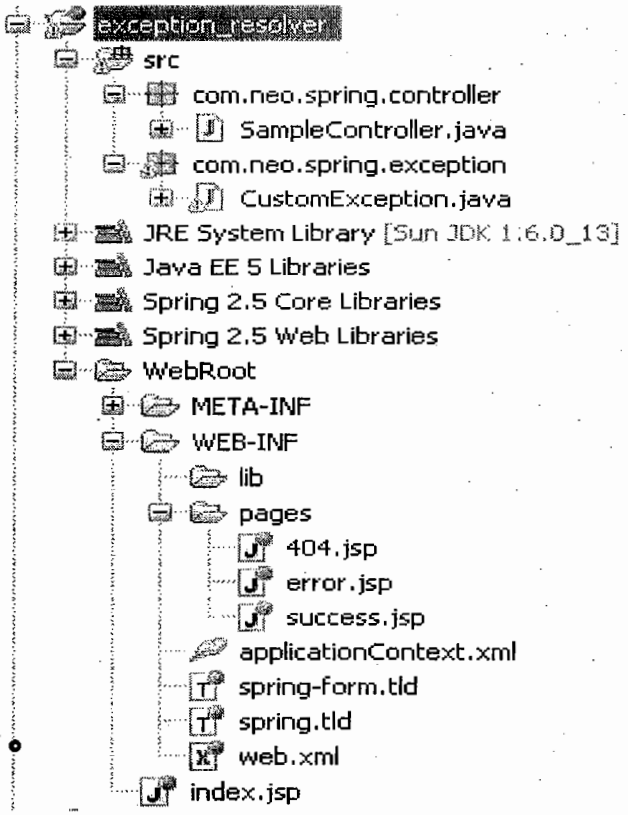
1. CustomException is throw, it will maps to the view name "error".
2. Any Exception or its sub class is throw, it will also maps to the view name "error".

With **InternalResourceViewResolver** is configured,

1. "error" will return "/WEB-INF/pages/error.jsp".

If given url is not correct then displays 404.jsp.

In the JSP page, we can access the exception instance via `<%= exception.getMessage() %>`



SampleController.java

```
1. package com.neo.spring.controller;
```

```
2.
3. import javax.servlet.http.HttpServletRequest;
4. import javax.servlet.http.HttpServletResponse;
5. import org.springframework.web.servlet.ModelAndView;
6. import org.springframework.web.servlet.mvc.AbstractController;
7. import com.neo.spring.exception.CustomException;
8.
9. public class SampleController extends AbstractController {
10.         @Override
11.         protected ModelAndView handleRequestInternal(HttpServletRequest
12.                 request, HttpServletResponse response) throws Exception {
13.               String outputPage = "error";
14.               String type = request.getParameter("type");
15.               if ("CustomException".equalsIgnoreCase(type)) {
16.                     throw new CustomException("user defined exception");
17.               } else if ("AIOB".equalsIgnoreCase(type)) {
18.                     throw new ArrayIndexOutOfBoundsException(
19.                     "testing error page for array index out of bounds");
20.               } else if ("success".equalsIgnoreCase(type)) {
21.                     outputPage = "success";
22.               }
23.               return new ModelAndView(outputPage);
24.         }
25.   }
```

CustomException.java

```
1. package com.neo.web.exception;
2. public class CustomException extends RuntimeException {
3.    public CustomException(String message) {
4.          super(message);
5.    }
6. }
```

web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
3.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5.    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
6.
7.    <servlet>
8.          <servlet-name>front-controller</servlet-name>
9.          <servlet-class>
10.              org.springframework.web.servlet.DispatcherServlet
11.         </servlet-class>
12.         <init-param>
13.               <param-name>contextConfigLocation</param-name>
14.               <param-value>/WEB-INF/applicationContext.xml</param-value>
15.         </init-param>
16.   </servlet>
17.   <servlet-mapping>
18.         <servlet-name>front-controller</servlet-name>
19.         <url-pattern>*.sekhar</url-pattern>
20.   </servlet-mapping>
21.   <listener>
```

```
22.         <listener-class>
23.              org.springframework.web.context.ContextLoaderListener
24.         </listener-class>
25.    </listener>
26.         <welcome-file-list>
27.               <welcome-file>index.jsp</welcome-file>
28.         </welcome-file-list>
29.    </web-app>
```

applicationContext.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.        xmlns:p="http://www.springframework.org/schema/p"
5.     xsi:schemaLocation="http://www.springframework.org/schema/beans
6.       http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
7.
8. <!-- Controllers configuration -->
9.      <bean id="sc" class="com.neo.spring.controller.SampleController" />
10.    <!-- Controllers configuration -->
11.
12.    <!-- Handler Mappings configuration-->
13.         <bean id="suhm"
14.          class="org.springframework.web.servlet.handler.SimpleUrlHandler
15.                                                                  Mapping">
16.               <property name="mappings">
17.                    <props>
18.                         <prop key="/sample.sekhar">sc</prop>
19.                    </props>
20.               </property>
21.         </bean>
22.    <!-- Handler Mappings configuration -->
23.
24.    <!-- View resolver configurations -->
25.          <bean id="irvr"
26.          class="org.springframework.web.servlet.view.InternalResourceView
27.                                                              Resolver">
28.               <property name="viewClass"
29.    value="org.springframework.web.servlet.view.InternalResourceView" />
30.               <property name="prefix" value="/WEB-INF/pages/" />
31.               <property name="suffix" value=".jsp" />
32.          </bean>
33.    <!-- View resolver configurations -->
34.
35.    <!-- Exception resolver configurations -->
36.          <bean
37.          class="org.springframework.web.servlet.handler.SimpleMapping
38.                                                   ExceptionResolver">
39.               <property name="exceptionMappings">
40.                    <props>
41.                       <prop key="com.neo.spring.exception.Custom
42.                                                   Exception">error</prop>
43.                       <prop key="java.lang.Exception">error</prop>
44.                    </props>
45.               </property>
```

```
46.         </bean>
47.         <!-- Exception resolver configurations -->
48.
49.   </beans>
```

index.jsp

```
1. <html>
2.   <body bgcolor="pink" >
3. <center>
4.    <h1>Error code demo application</h1>
5.    <hr/>
6.    <form action="./sample.sekhar">
7.    Type :
8.    <select name="type" >
9.          <option value="CustomException" >CustomException</option>
10.               <option value="AIOB" >ArrayIndexOutOfBoundsException
11.               </option>
12.               <option value="success" >success</option>
13.         </select> <br/>
14.         <input type="submit" value="Test Me" >
15.         </form>
16.     </center>
17.     </body>
18.   </html>
```

error.jsp

```
1. <%@page isErrorPage="true" %>
2. <html>
3.   <body bgcolor="pink" >
4.    <center>
5.    <h1>Error code demo application</h1>
6.    <hr/>
7.    <h2><%= exception.getMessage() %></h2>
8.    </center>
9.   </body>
10.   </html>
```

404.jsp

```
1. <html>
2.   <body bgcolor="pink" >
3. <center>
4.    <h1>Error code demo application</h1>
5.    <hr/>
6.    <h2>The request url is not correct. Pls try with correct url</h2>
7. </center>
8.   </body>
9. </html >
```

success.jsp

```
1. <%@page isErrorPage="true" %>
2. <html>
3.   <body bgcolor="pink" >
4. <center>
5.    <h1>Error code demo application</h1>
6.    <hr/>
```

```
7.    <h2>success</h2>
8. </center>
9.   </body>
10.   </html>
```

Note
If both Spring and servlet container are configured to handle the **java.lang.Exception**, the Spring will has more high priority to handle it.

Spring bean definition

```
<bean class="org.springframework.web.servlet.handler.SimpleMapping
                                                        ExceptionResolver">
       <property name="exceptionMappings">
        <props>
                <prop key="java.lang.Exception">error</prop>
        </props>
        </property>
</bean>
```

web.xml

```
<error-page>
    <exception-type>java.lang.Exception</exception-type>
        <location>/WEB-INF/pages/error.jsp</location>
</error-page>
```

InternalResourceViewResolver

In Spring MVC application, **InternalResourceViewResolver** class is used to resolve the internal resource view based on a predefined URL pattern, which is adding a predefined prefix and suffix to the view name (**prefix + view name + suffix**), and generate the final view page URL.

What's internal resource views?
In Spring MVC application, or good practice, it's always recommended to put all the views or JSP files under WEB-INF folder, to protect it from the direct access via manual entered URL. Those views under WEB-INF folder are named as internal resource views, as it's only accessible by the servlet or Spring's controllers class.

Controller
A controller class to return a view, named "**welcomePage**".

```
//...
public class WelcomeController extends AbstractController{

        @Override
        protected ModelAndView handleRequestInternal(HttpServletRequest
                    request, HttpServletResponse response) throws Exception {

                ModelAndView model = new ModelAndView("welcomePage");
                return model;
        }
}
```

InternalResourceViewResolver

Register the **InternalResourceViewResolver** bean in the Spring's bean configuration file.

```
<beans ---------------->

                <!-- Handler  Configuration -->

                <!-- Controller Configuration -->

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

Now, Spring will resolve the view's name "**WelcomePage**" in the following way :
prefix + view name + suffix = /WEB-INF/pages/**WelcomPage**.jsp

Finally, with a view name "**welcomePage**" returned by controller, the InternalResourceViewResolver will return an URL "**/WEB-INF/pages/welcomePage.jsp**" back to the DispatcherServlet.

XmlViewResolver

In Spring MVC application, **XmlViewResolver** class is used to resolve the view based on view beans in the XML file. By default, XmlViewResolver will loads the view beans from **/WEB-INF/views.xml**, however, this location can be overridden through the "**location**" property.

```
<beans ...>
        <bean class="org.springframework.web.servlet.view.XmlViewResolver">
           <property name="location">
                <value>/WEB-INF/spring-views.xml</value>
           </property>
        </bean>
</beans>
```

In above case, it loads the view beans from "**/WEB-INF/spring-views.xml**". See XmlViewResolver example :

1. Controller
A controller class to return a view, named "**welcomePage**".

```
/...
public class WelcomeController extends AbstractController{

        @Override
        protected ModelAndView handleRequestInternal(HttpServletRequest
                        request,HttpServletResponse response) throws Exception {

                ModelAndView model = new ModelAndView("welcomePage");
                return model;
        }
}
```

2. XmlViewResolver

Register the XmlViewResolver in the Spring's bean configuration file, loads the view beans from "**/WEB-INF/spring-views.xml**".

```
<beans ---------------->

                <!-- Handler  Configuration -->

                <!-- Controller Configuration -->

        <bean class="org.springframework.web.servlet.view.XmlViewResolver">
           <property name="location">
               <value>/WEB-INF/spring-views.xml</value>
           </property>
        </bean>
</beans>
```

3. View beans

The "**view bean**" is just a normal bean declared in the Spring's bean configuration file, just declare each view as a normal Spring's bean, where

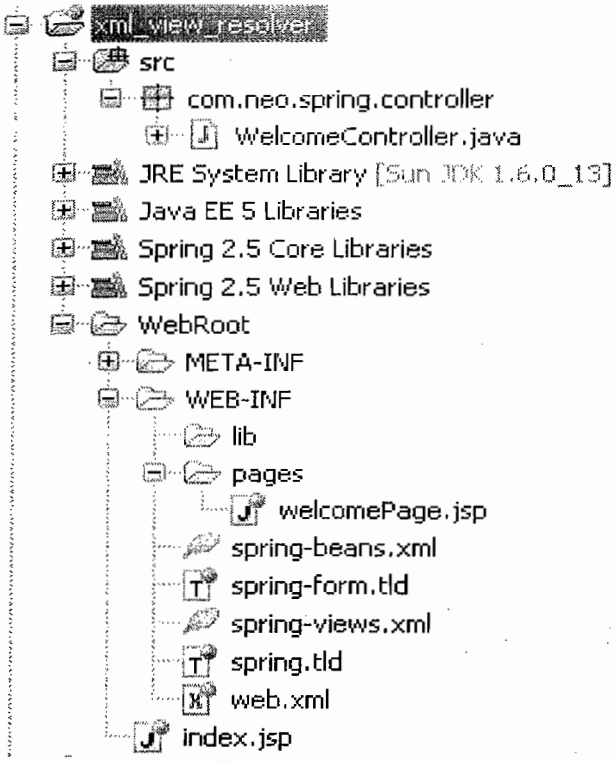
1. "**id**" is the "view name" to resolve.
2. "**class**" is the type of the view.
3. "**url**" property is the view's url location.

spring-views.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-2.5.xsd">

        <bean id="welcomePage"
           class="org.springframework.web.servlet.view.JstlView">
                <property name="url" value="/WEB-INF/pages/welcomePage.jsp" />
        </bean>

</beans>
```

4. How it work?

Finally, with a view name "**welcomePage**" returned by controller, the XmlViewResolver will find the bean id "**welcomePage**" in **spring-views.xml** file, and return the corresponds view's URL "**/WEB-INF/pages/welcomePage.jsp**" back to the DispatcherServlet.



WelcomeController.java

```
1. package com.neo.spring.controller;
2.
3. import javax.servlet.http.HttpServletRequest;
4. import javax.servlet.http.HttpServletResponse;
5. import org.springframework.web.servlet.ModelAndView;
6. import org.springframework.web.servlet.mvc.AbstractController;
7.
8. public class WelcomeController extends AbstractController {
9.    @Override
10.         protected ModelAndView handleRequestInternal(HttpServletRequest
11.               request, HttpServletResponse response) throws Exception {
12.               String name = request.getParameter("name");
13.               String welcomeMessage = null;
14.               if (name != null && name.trim().length() > 0) {
15.                     welcomeMessage = "Welcome " + name;
16.               } else {
17.                     welcomeMessage = "Welcome Guest";
18.               }
19.               return new ModelAndView("welcomePage", "welcomeMsg",
20.                                                      welcomeMessage);
21.         }
22.   }
```

web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
3.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5.    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
6.
```

```
7.    <servlet>
8.          <servlet-name>front-controller</servlet-name>
9.          <servlet-class>
10.                org.springframework.web.servlet.DispatcherServlet
11.         </servlet-class>
12.               <init-param>
13.                     <param-name>contextConfigLocation</param-name>
14.                     <param-value>/WEB-INF/spring-beans.xml</param-value>
15.               </init-param>
16.         </servlet>
17.         <servlet-mapping>
18.               <servlet-name>front-controller</servlet-name>
19.               <url-pattern>*.sekhar</url-pattern>
20.         </servlet-mapping>
21.         <welcome-file-list>
22.               <welcome-file>index.jsp</welcome-file>
23.         </welcome-file-list>
24.   </web-app>
```

spring-beans.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xmlns:p="http://www.springframework.org/schema/p"
5.     xsi:schemaLocation="http://www.springframework.org/schema/beans
6.     http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
7.
8.     <!-- Controllers configuration -->
9.     <bean id="wc" class="com.neo.spring.controller.WelcomeController" />
10.          <!-- Controllers configuration -->
11.
12.          <!-- Handler Mappings configuration-->
13.          <bean id="suhm" class="org.springframework.web.servlet.handler
14.                                                .SimpleUrlHandlerMapping">
15.               <property name="mappings">
16.                    <props>
17.                         <prop key="/welcome.sekhar">wc</prop>
18.                    </props>
19.               </property>
20.          </bean>
21.          <!-- Handler Mappings configuration -->
22.
23.          <!-- View resolver configurations -->
24.          <bean
25.           class="org.springframework.web.servlet.view.XmlViewResolver">
26.               <property name="location">
27.                    <value>/WEB-INF/spring-views.xml</value>
28.               </property>
29.          </bean>
30.
31.          <!-- View resolver configurations -->
32.
33.    </beans>
```

spring-views.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans
3.     xmlns="http://www.springframework.org/schema/beans"
4.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5.     xmlns:p="http://www.springframework.org/schema/p"
6.     xsi:schemaLocation="http://www.springframework.org/schema/beans
7.     http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
8.
9.     <bean id="welcomePage"
10.          class="org.springframework.web.servlet.view.JstlView">
11.          <property name="url" value="/WEB-INF/pages/welcomePage.jsp" />
12.    </bean>
13.
14.    </beans>
```

index.jsp

```
1. <html>
2.    <body bgcolor="pink" >
3.      <center>
4.           <h1>XmlViewReslover demo application</h1>
5.           <hr>
6.           <form action="./welcome.sekhar" method="post" >
7.                Name : <input type="text" name="name"  > <br/>
8.                <input type="submit" value="Get Welcome" >
9.           </form>
10.       </center>
11.     </body>
12.   </html>
```

welcomePage.jsp

```
1. <%@page isErrorPage="true" %>
2. <html>
3.    <body bgcolor="pink" >
4. <center>
5.     <h1>XmlViewReslover demo application</h1>
6.     <hr/>
7.     <h2>${welcomeMsg}</h2>
8. </center>
9.    </body>
10.    </html>
```

ResourceBundleViewResolver

In Spring MVC application, **ResourceBundleViewResolver** class is used to resolve the view based on view beans in the properties file. By default, ResourceBundleViewResolver will loads the view beans from **views.properties**, which located at the root of the project class path. However, this location can be overridden through the "**basename**" property.

```
<beans ...>
        <bean class="org.springframework.web.servlet.view.ResourceBundleViewResolver">
                <property name="basename" value="spring-views" />
        </bean>
</beans>
```

In above case, it loads the view beans from "**spring-views.properties**", which located at the root of the project class path.

Note
The ResourceBundleViewResolver has the ability to load view beans from different resource bundles for different locales, but this use case is rarely required.

See ResourceBundleViewResolver example:

1. Controller
A controller class to return a view, named "**welcomePage**".

```
//...
public class WelcomeController extends AbstractController{

        @Override
        protected ModelAndView handleRequestInternal(HttpServletRequest
                    request, HttpServletResponse response) throws Exception {

                ModelAndView model = new ModelAndView("welcomePage");
                return model;
        }
}
```

2. ResourceBundleViewResolver
Register the ResourceBundleViewResolver in the Spring's bean configuration file, change the default view beans location to "**spring-views.properties**".

```
<beans ---------------->

                <!-- Handler  Configuration -->

                <!-- Controller Configuration -->

        <bean class="org.springframework.web.servlet.view.Resource
                                                        BundleViewResolver">
                <property name="basename" value="spring-views" />
        </bean>
</beans>
```

3. View beans
Declare each view in a normal resource bundle style (key & message), where

1. "**welcomePage**" is the view name to match.
2. "**.(class)**" is the type of view.
3. "**.url**" is the view's URL location.

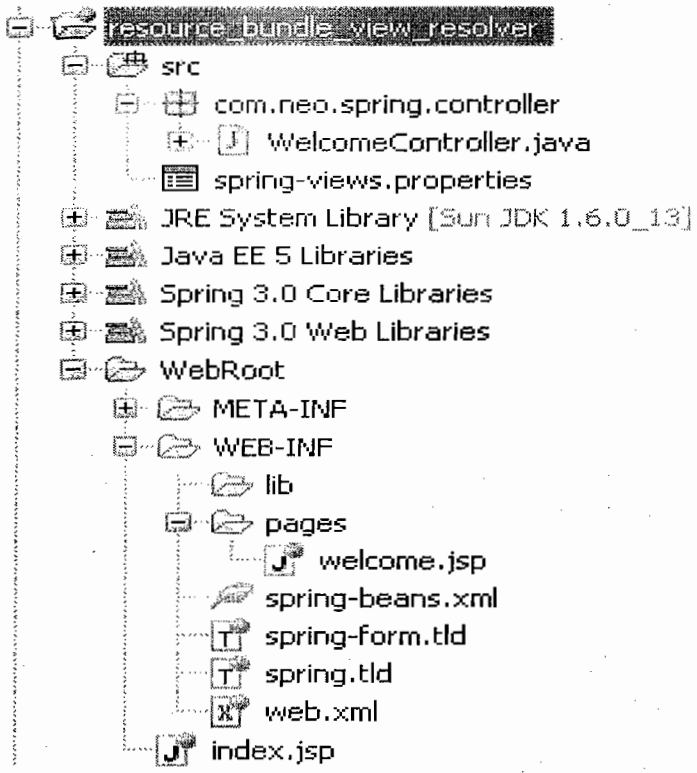
spring-views.properties

```
welcomePage.(class)=org.springframework.web.servlet.view.JstlView
welcomePage.url=/WEB-INF/pages/welcome.jsp
```

Note
Put this "spring-views.properties" file into the root of the project class path.

4. How it work?

Finally, with a view name "**welcomePage**" returned by controller, the ResourceBundleViewResolver will find the key start with "**welcomePage**" in "**spring-views.properties**" file, and return the corresponds view's URL "**/WEB-INF/pages/welcome.jsp**" back to the DispatcherServlet.



WelcomeController.java

```
1. package com.neo.spring.controller;
2.
3. import javax.servlet.http.HttpServletRequest;
4. import javax.servlet.http.HttpServletResponse;
5. import org.springframework.web.servlet.ModelAndView;
6. import org.springframework.web.servlet.mvc.AbstractController;
7.
8. public class WelcomeController extends AbstractController {
9.     @Override
10.         protected ModelAndView handleRequestInternal(HttpServletRequest
11.                 request, HttpServletResponse response) throws Exception {
12.                 String name = request.getParameter("name");
13.                 String welcomeMessage = null;
14.                 if (name != null && name.trim().length() > 0) {
15.                         welcomeMessage = "Welcome " + name;
16.                 } else {
17.                         welcomeMessage = "Welcome Guest";
18.                 }
19.                 return new ModelAndView("welcomePage", "welcomeMsg",
20.                                                         welcomeMessage);
21.         }
22.   }
```

spring-views.properties

```
1. welcomePage.(class)=org.springframework.web.servlet.view.JstlView
```

```
2. welcomePage.url=/WEB-INF/pages/welcome.jsp
```

web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5.     http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
6.
7.     <servlet>
8.           <servlet-name>front-controller</servlet-name>
9.           <servlet-class>
10.                org.springframework.web.servlet.DispatcherServlet
11.          </servlet-class>
12.                <init-param>
13.                     <param-name>contextConfigLocation</param-name>
14.                     <param-value>/WEB-INF/spring-beans.xml</param-value>
15.                </init-param>
16.          </servlet>
17.          <servlet-mapping>
18.                <servlet-name>front-controller</servlet-name>
19.                <url-pattern>*.sekhar</url-pattern>
20.          </servlet-mapping>
21.          <welcome-file-list>
22.                <welcome-file>index.jsp</welcome-file>
23.          </welcome-file-list>
24.    </web-app>
```

spring-beans.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xmlns:p="http://www.springframework.org/schema/p"
5.     xsi:schemaLocation="http://www.springframework.org/schema/beans
6.     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7.
8.     <!-- Controllers configuration -->
9.     <bean id="wc" class="com.neo.spring.controller.WelcomeController" />
10.          <!-- Controllers configuration -->
11.
12.          <!-- Handler Mappings configuration-->
13.          <bean id="suhm" class="org.springframework.web.servlet.handler
14.                                              .SimpleUrlHandlerMapping">
15.                <property name="mappings">
16.                     <props>
17.                          <prop key="/welcome.sekhar">wc</prop>
18.                     </props>
19.                </property>
20.          </bean>
21.          <!-- Handler Mappings configuration -->
22.
23.          <!-- View resolver configurations -->
24.
25.          <bean class="org.springframework.web.servlet.view.ResourceBundle
26.                                                        ViewResolver">
```

```
27.             <property name="basename" value="spring-views" />
28.         </bean>
29.
30.         <!-- View resolver configurations -->
31.
32.   </beans>
```

index.jsp

```
1. <html>
2.   <body bgcolor="pink" >
3.     <center>
4.         <h1>ResourceBundleViewReslover demo application</h1>
5.         <hr>
6.         <form action="./welcome.sekhar" method="post" >
7.             Name : <input type="text" name="name"  > <br/>
8.             <input type="submit" value="Get Welcome" >
9.         </form>
10.      </center>
11.    </body>
12.  </html>
```

welcome.jsp

```
1. <%@page isErrorPage="true" %>
2. <html>
3.   <body bgcolor="pink" >
4. <center>
5.    <h1>ResourceBundleViewReslover demo application</h1>
6.    <hr/>
7.    <h2>${welcomeMsg}</h2>
8. </center>
9.   </body>
10.  </html>
```

Bean Name View Resolver

Bean Name View Resolver which will dynamically generate View in jsp or Pdf or Excel Formats. This resolver can be handy for small applications, keeping all definitions ranging from controllers to views in the same place i.e., in spring configuration itself.

Controller

A controller class to return a view, named "**welcomePage**".

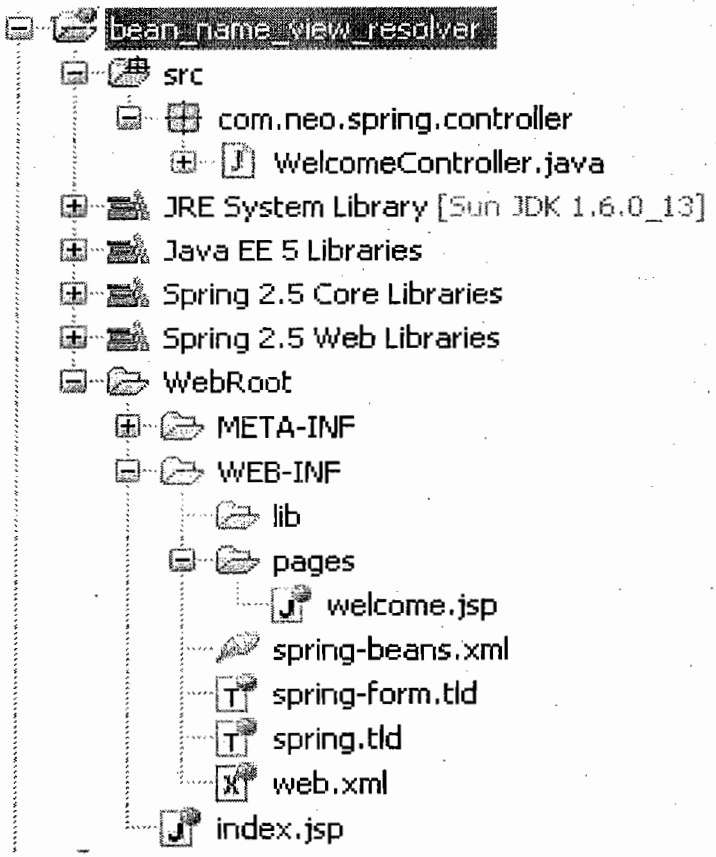
```
//...
public class WelcomeController extends AbstractController{

        @Override
        protected ModelAndView handleRequestInternal(HttpServletRequest
                    request, HttpServletResponse response) throws Exception {

                ModelAndView model = new ModelAndView("welcomePage");
                return model;
        }
}
```

BeanNameViewResolver

Register the BeanNameViewResolver in the Spring's bean configuration file.

```
<beans -------------->

                <!-- Handler  Configuration -->

                <!-- Controller Configuration -->

        <bean id="viewresolver" class="org.springframework.web.servlet.view.
                                                        BeanNameViewResolver"/>
         <bean id="welcomePage" class="org.springframework.web.servlet.view.
                                                                JstlView"/>
                 <property name="url" value="/WEB-INF/pages/welcome.jsp" />
        </bean>
</beans>
```

Finally, with a view name "**welcomePage**" returned by controller, the BeanNameViewResolver will find the key start with "**welcomePage**" in spring configuration file itself, and return the corresponds view's URL "**/WEB-INF/pages/welcome.jsp**" back to the DispatcherServlet.



WelcomeController.java

```
1. package com.neo.spring.controller;
2.
3. import javax.servlet.http.HttpServletRequest;
4. import javax.servlet.http.HttpServletResponse;
5. import org.springframework.web.servlet.ModelAndView;
6. import org.springframework.web.servlet.mvc.AbstractController;
7.
```

```
8. public class WelcomeController extends AbstractController {
9.     @Override
10.         protected ModelAndView handleRequestInternal(HttpServletRequest
11.                 request, HttpServletResponse response) throws Exception {
12.                 String name = request.getParameter("name");
13.                 String welcomeMessage = null;
14.                 if (name != null && name.trim().length() > 0) {
15.                         welcomeMessage = "Welcome " + name;
16.                 } else {
17.                         welcomeMessage = "Welcome Guest";
18.                 }
19.                 return new ModelAndView("welcomePage", "welcomeMsg",
20.                                                                 welcomeMessage);
21.         }
22.   }
```

web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5.     http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
6.
7.     <servlet>
8.           <servlet-name>front-controller</servlet-name>
9.           <servlet-class>
10.                  org.springframework.web.servlet.DispatcherServlet
11.          </servlet-class>
12.                <init-param>
13.                      <param-name>contextConfigLocation</param-name>
14.                      <param-value>/WEB-INF/spring-beans.xml</param-value>
15.                </init-param>
16.          </servlet>
17.          <servlet-mapping>
18.                <servlet-name>front-controller</servlet-name>
19.                <url-pattern>*.sekhar</url-pattern>
20.          </servlet-mapping>
21.          <welcome-file-list>
22.                <welcome-file>index.jsp</welcome-file>
23.          </welcome-file-list>
24.   </web-app>
```

spring-beans.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xmlns:p="http://www.springframework.org/schema/p"
5.     xsi:schemaLocation="http://www.springframework.org/schema/beans
6.     http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
7.
8.     <!-- Controllers configuration -->
9.     <bean id="wc" class="com.neo.spring.controller.WelcomeController" />
10.          <!-- Controllers configuration -->
11.
12.          <!-- Handler Mappings configuration-->
```

```
13.         <bean id="suhm"class="org.springframework.web.servlet.handler
14.                                                 .SimpleUrlHandlerMapping">
15.               <property name="mappings">
16.                     <props>
17.                           <prop key="/welcome.sekhar">wc</prop>
18.                     </props>
19.               </property>
20.         </bean>
21.         <!-- Handler Mappings configuration -->
22.
23.         <!-- View resolver configurations -->
24.
25.    <bean id="viewresolver" class="org.springframework.web.servlet.view
26.                                                 .BeanNameViewResolver" />
27.
28.         <!-- View resolver configurations -->
29.
30.         <!-- View beans configuration -->
31.
32.    <bean id="welcomePage" class="org.springframework.web.servlet.view
33.                                                         .JstlView">
34.               <property name="url" value="/WEB-INF/pages/welcome.jsp" />
35.         </bean>
36.
37.         <!-- View beans configuration -->
38.    </beans>
```

index.jsp

```
1. <html>
2.    <body bgcolor="pink" >
3.      <center>
4.           <h1>BeanNameViewReslover demo application</hl>
5.           <hr>
6.           <form action="./welcome.sekhar" method="post" >
7.                Name : <input type="text" name="name"  > <br/>
8.                <input type="submit" value="Get Welcome" >
9.           </form>
10.       </center>
11.     </body>
12.   </html>
```

welcome.jsp

```
1. </html>
2. <%@page isErrorPage="true" %>
3. <html>
4.    <body bgcolor="pink" >
5. <center>
6.     <h1>BeanNameViewReslover demo application</h1>
7.     <hr/>
8.     <h2>${welcomeMsg}</h2>
9. </center>
10.      </body>
11.   </html>
```

Configure multiple view resolvers priority in Spring MVC

Problem

In Spring MVC application, often times, we may applying few view resolver strategies to resolve the view name.

For example, combine three view resolvers together i.e., **InternalResourceViewResolver**, **ResourceBundleViewResolver** and **XmlViewResolver**.

```
<beans ...>
        <bean class="org.springframework.web.servlet.view.XmlViewResolver">
              <property name="location">
                 <value>/WEB-INF/spring-views.xml</value>
              </property>
        </bean>

        <bean class="org.springframework.web.servlet.view.Resource
                                                          BundleViewResolver">
              <property name="basename" value="spring-views" />
        </bean>

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

But, if a view name is returned, which view resolver strategy will be used?

Solution

If multiple view resolver strategies are applied, we have to declare the priority through "**order**" property, where the **lower order value has a higher priority**, for example:

```
<beans ...>
        <bean class="org.springframework.web.servlet.view.XmlViewResolver">
             <property name="location">
                <value>/WEB-INF/spring-views.xml</value>
             </property>
             <property name="order" value="0" />
        </bean>

        <bean class="org.springframework.web.servlet.view.ResourceBundleViewResolver">
             <property name="basename" value="spring-views" />
             <property name="order" value="1" />
        </bean>

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

```
            </property>
            <property name="order" value="2" />
        </bean>
</beans>
```

Now, if a view name is returned, the view resolving strategy works in the following order:

`XmlViewResolver --> ResourceBundleViewResolver --> InternalResourceViewResolver`

Note
The **InternalResourceViewResolver** must always **assign with the lowest priority** (largest order number), because it will resolve the view no matter what view name is returned. It caused other view resolvers have no chance to resolve the view if they have lower priority.

AbstractPdfView

Spring MVC comes with **AbstractPdfView** class to export data to pdf file via Bruno Lowagie's **iText** library.
This example shows the use of **AbstractPdfView** class in Spring MVC application to export data to pdf file for download.

Controller
A controller class, generate dummy data for demonstration, and get the request parameter to determine which view to return. If the request parameter is equal to "**PDF**", then return a Pdf view (**AbstractPdfView**).

AccountController.java

```
//...

public class AccountController extends AbstractController{

        @Override
        protected ModelAndView handleRequestInternal(HttpServletRequest
                        request,HttpServletResponse response) throws Exception {

       // Here we are calling the method on service class
              return new ModelAndView("accountPdfView", "accounts", accounts);

        }
}
```

AccountDetailsPdfView
Create a pdf view by extending the **AbstractPdfView** class, override **buildPdfDocument()** method to populate the data to pdf file.
The **AbstractPdfView** is using the **iText API** to generate the pdf file.

AccountDetailsPdfView.java

```
/...

public class AccountDetailsPdfView extends AbstractPdfView{

        @Override
        protected void buildPdfDocument(Map model, Document document,
                PdfWriter writer, HttpServletRequest request,
                HttpServletResponse response) throws Exception {
 //Here we get the account details from model and checking the condition and
 //Displaying the Account details.

        }
}
```

Spring Configuration
Create a **XmlViewResolver** for the Pdf view.

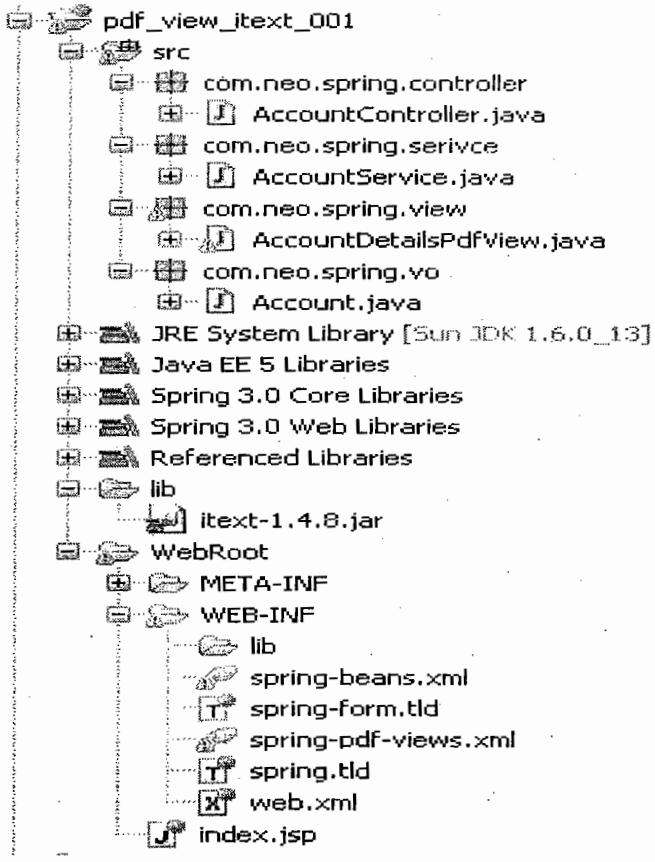
```
<beans ...>
                <!-- Handler  Configuration -->

                <!-- Controller Configuration -->

    <bean id="vr"
          class="org.springframework.web.servlet.view.XmlViewResolver">
       <property name="location" value="/WEB-INF/spring-pdf-views.xml" />
```

```
    </bean>

</beans>
```

spring-pdf-views.xml

```
<beans ...">

   <bean id="accountPdfView"
         class="com.neo.spring.view.AccountDetailsPdfView">
   </bean>
</beans>
```

Now we will see complete application:



Account.java

```
1. package com.neo.spring.vo;
2. public class Account {
3.     private int accno;
4.     private String name;
5.     private double balance;
6.
7.     public int getAccno() {
8.           return accno;
9.     }
10.
11.          public void setAccno(int accno) {
12.                this.accno = accno;
```

```
13.         }
14.
15.         public String getName() {
16.               return name;
17.         }
18.
19.         public void setName(String name) {
20.               this.name = name;
21.         }
22.
23.         public double getBalance() {
24.               return balance;
25.         }
26.
27.         public void setBalance(double balance) {
28.               this.balance = balance;
29.         }
30.   }
```

AccountController.java

```
1. package com.neo.spring.controller;
2.
3. import java.util.List;
4. import javax.servlet.http.HttpServletRequest;
5. import javax.servlet.http.HttpServletResponse;
6. import org.springframework.web.servlet.ModelAndView;
7. import org.springframework.web.servlet.mvc.AbstractController;
8. import com.neo.spring.serivce.AccountService;
9. import com.neo.spring.vo.Account;
10.
11.   public class AccountController extends AbstractController {
12.         private AccountService service;
13.
14.         public void setService(AccountService service) {
15.               this.service = service;
16.         }
17.
18.         @Override
19.         protected ModelAndView handleRequestInternal(HttpServletRequest
20.                   request, HttpServletResponse response) throws Exception {
21.               List<Account> accounts = service.getAccounts();
22.
23.               return new ModelAndView("accountPdfView", "accounts",
24.                                                                   accounts);
25.         }
26.   }
```

AccountService.java

```
1. package com.neo.spring.serivce;
2.
3. import java.util.ArrayList;
4. import java.util.List;
5. import com.neo.spring.vo.Account;
6.
7. public class AccountService {
```

```
8.    public List<Account> getAccounts()  {
9.          List<Account> accounts = new ArrayList<Account>();
10.               Account account = null;
11.               for(int i=0; i<10; i++){
12.                     account= new Account();
13.                     account.setAccno(i);
14.                     account.setName("Sekhar-"+i);
15.                     account.setBalance(i*10);
16.                     accounts.add(account);
17.               }
18.               return accounts;
19.         }
20.   }
```

AccountDetailsPdfView.java

```
1. package com.neo.spring.view;
2.
3. import java.util.List;
4. import java.util.Map;
5. import javax.servlet.http.HttpServletRequest;
6. import javax.servlet.http.HttpServletResponse;
7. import org.springframework.web.servlet.view.document.AbstractPdfView;
8. import com.lowagie.text.Document;
9. import com.lowagie.text.Paragraph;
10.   import com.lowagie.text.Table;
11.   import com.lowagie.text.pdf.PdfWriter;
12.   import com.neo.spring.vo.Account;
13.
14.   public class AccountDetailsPdfView extends AbstractPdfView {
15.         @Override
16.         protected void buildPdfDocument(Map<String, Object> model,
17.                     Document document, PdfWriter writer,
18.                     HttpServletRequest request,
19.                     HttpServletResponse response) throws Exception {
20.               List<Account> accounts = (List<Account>)
21.                                                   model.get("accounts");
22.               Paragraph paragraph = null;
23.               if (accounts == null || accounts.isEmpty()) {
24.                   paragraph = new Paragraph("No ACCOUNT DETAILS FOUND ");
25.                     document.add(paragraph);
26.               }
27.
28.               paragraph = new Paragraph("Account Details");
29.               paragraph.setAlignment("center");
30.               document.add(paragraph);
31.
32.               Table table = new Table(3);
33.               table.addCell("ACCNO");
34.               table.addCell("NAME");
35.               table.addCell("BALANCE");
36.
37.               for (Account account : accounts) {
38.                     table.addCell(account.getAccno() + "");
39.                     table.addCell(account.getName());
40.                     table.addCell(account.getBalance() + "");
```

```
41.                 }
42.                 document.add(table);
43.           }
44.     }
```

spring-beans.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xmlns:p="http://www.springframework.org/schema/p"
5.     xsi:schemaLocation="http://www.springframework.org/schema/beans
6.     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7.
8.     <!-- Services configuration -->
9.     <bean id="as" class="com.neo.spring.serivce.AccountService" />
10.          <!-- Services configuration -->
11.
12.          <!-- Controllers configuration -->
13.         <bean id="ac" class="com.neo.spring.controller.AccountController">
14.               <property name="service" ref="as" />
15.          </bean>
16.          <!-- Controllers configuration -->
17.
18.          <!-- Handler Mappings configuration-->
19.          <bean id="hm" class="org.springframework.web.servlet.handler
20.                                                  .SimpleUrlHandlerMapping">
21.               <property name="mappings">
22.                     <props>
23.                           <prop key="/accountsPdf.sekhar">ac</prop>
24.                     </props>
25.               </property>
26.          </bean>
27.          <!-- Handler Mappings configuration -->
28.
29.          <!-- View resolver configurations -->
30.          <bean id="vr" class="org.springframework.web.servlet.view.Xml
31.                                                          ViewResolver">
32.               <property name="location" value="/WEB-INF/spring-pdf-
33.                                                          views.xml" />
34.          </bean>
35.          <!-- View resolver configurations -->
36.
37.    </beans>
```

spring-pdf-views.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xmlns:p="http://www.springframework.org/schema/p"
5.     xsi:schemaLocation="http://www.springframework.org/schema/beans
6.     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7.
8.     <bean id="accountPdfView"
9.          class="com.neo.spring.view.AccountDetailsPdfView" />
10.
```

```
11.   </beans>
```

web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5.     http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
6.
7.     <servlet>
8.           <servlet-name>front-controller</servlet-name>
9.     <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
10.              <init-param>
11.                    <param-name>contextConfigLocation</param-name>
12.                    <param-value>/WEB-INF/spring-beans.xml</param-value>
13.              </init-param>
14.          </servlet>
15.          <servlet-mapping>
16.                <servlet-name>front-controller</servlet-name>
17.                <url-pattern>*.sekhar</url-pattern>
18.          </servlet-mapping>
19.          <welcome-file-list>
20.                <welcome-file>index.jsp</welcome-file>
21.          </welcome-file-list>
22.    </web-app>
```

index.jsp

```
1. <html>
2.    <body bgcolor="pink" >
3.      <center>
4.           <h1>Pdf view demo application</h1>
5.           <hr>
6.           <h3> <a  href="./accountsPdf.sekhar" >Get Account
7.                                                  details</a></h3>
8.      </center>
9.      </body>
10.   </html>
```

AbstractExcelView

Spring MVC comes with **AbstractExcelView** class to export data to Excel file via **Apache POI** library. This example shows the use of **AbstractExcelView** class in Spring MVC application to export data to Excel file for download.

1. Apache POI

Get the Apache POI library to create the excel file.

2. Controller

A controller class, generate dummy data for demonstration, and get the request parameter to determine which view to return. If the request parameter is equal to "EXCEL", then return an Excel view (**AbstractExcelView**).

AccountController.java

```
//...

public class AccountController extends AbstractController{
```

```
        @Override
        protected ModelAndView handleRequestInternal(HttpServletRequest
                    request,HttpServletResponse response) throws Exception {

       // Here we are calling the method on service class
             return new ModelAndView("accountDetailsExcelView", "accounts",
                                                                  accounts);
        }
}
```

3. AbstractExcelView

Create an Excel view by extends the **AbstractExcelView** class, and override the **buildExcelDocument()** method to populate the data to Excel file. The **AbstractExcelView** is using the **Apache POI API** to create the Excel file detail.

AccountDetailsExcelView.java

```
//...
public class AccountDetailsExcelView extends AbstractExcelView{

        @Override
        protected void buildExcelDocument(Map model, HSSFWorkbook workbook,
                HttpServletRequest request, HttpServletResponse response)
                throws Exception {
 //Here we get the account details from model and checking the condition and
 //Displaying the Account details.

        }
}
```

4. Spring Configuration

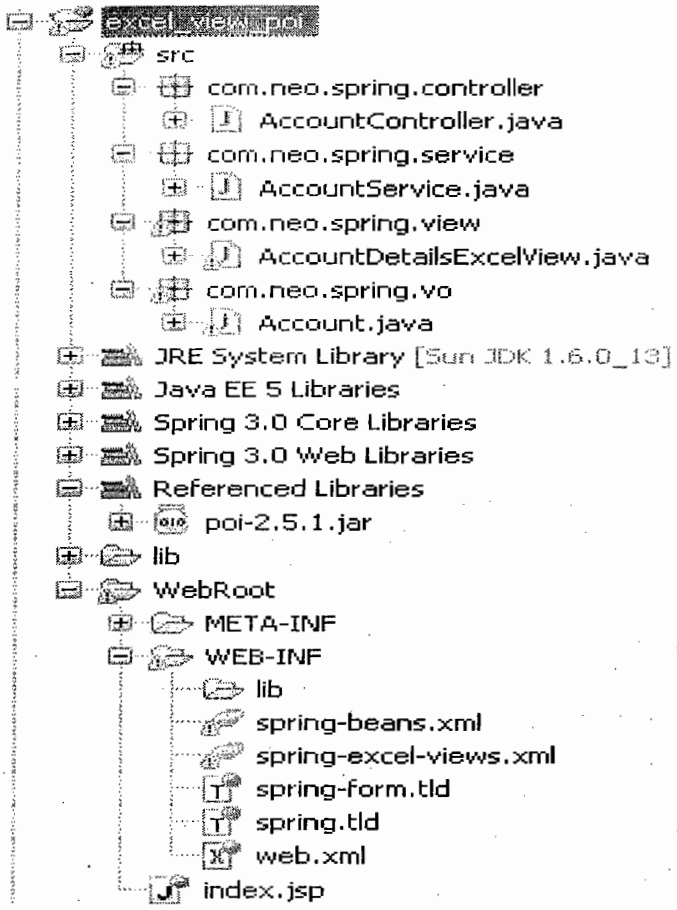
Create a **XmlViewResolver** for the Excel view.

```
<beans ...>
                <!-- Handler  Configuration -->

                <!-- Controller Configuration -->

    <bean class="org.springframework.web.servlet.view.XmlViewResolver">
       <property name="location" value="/WEB-INF/spring-excel-views.xml" />
    </bean>

</beans>
```

spring-excel-views.xml

```
<beans ...">

   <bean id="accountDetailsExcelView"
         class="com.neo.spring.view.AccountDetailsExcelView">
   </bean>
</beans>
```

Now we will see complete application:



Account.java

```
1. package com.neo.spring.vo;
2. public class  Account {
3.     private int accno;
4.     private String name;
5.     private float balance;
6.     public void setAccno(int accno)
7.     {
8.           this.accno=accno;
9.     }
10.    public  int getAccno()
11.    {
12.          return accno;
13.    }
14.    public  void setName(String name)
15.    {
16.          this.name=name;
17.    }
18.    public  String getName()
19.    {
20.          return name;
21.    }
22.    public void setBalance(float balance)
```

```
23.   {
24.         this.balance=balance;
25.   }
26.   public float getBalance()
27.   {
28.         return balance;
29.   }
30.   }
```

AccountController.java

```
1. package com.neo.spring.controller;
2.
3. import java.util.List;
4. import javax.servlet.http.HttpServletRequest;
5. import javax.servlet.http.HttpServletResponse;
6. import org.springframework.web.servlet.ModelAndView;
7. import org.springframework.web.servlet.mvc.AbstractController;
8. import com.neo.spring.service.AccountService;
9. import com.neo.spring.vo.Account;
10.
11.   public class AccountController extends AbstractController {
12.         private AccountService service;
13.
14.         public void setService(AccountService service) {
15.               this.service = service;
16.         }
17.
18.         @Override
19.         protected ModelAndView handleRequestInternal(HttpServletRequest
20.                   request, HttpServletResponse response) throws Exception {
21.               List<Account> accounts = service.getAccounts();
22.               return new ModelAndView("accountDetailsExcelView",
23.                                                   "accounts", accounts);
24.         }
25.   }
```

AccountService.java

```
1. package com.neo.spring.service;
2.
3. import java.util.ArrayList;
4. import java.util.List;
5. import com.neo.spring.vo.Account;
6.
7. public class AccountService {
8.    public List<Account> getAccounts() {
9.          List<Account> accounts = new ArrayList<Account>();
10.               Account account = null;
11.               for (int i = 0; i < 10; i++) {
12.                     account = new Account();
13.                     account.setAccno(i);
14.                     account.setName("Sekhar-" + i);
15.                     account.setBalance(i * 10);
16.                     accounts.add(account);
17.               }
18.               return accounts;
```

```
19.         }
20.   }
```

AccountDetailsExcelView.java

```
1. package com.neo.spring.view;
2.
3. import java.util.List;
4. import java.util.Map;
5. import javax.servlet.http.HttpServletRequest;
6. import javax.servlet.http.HttpServletResponse;
7. import org.apache.poi.hssf.usermodel.HSSFRow;
8. import org.apache.poi.hssf.usermodel.HSSFSheet;
9. import org.apache.poi.hssf.usermodel.HSSFWorkbook;
10.   import org.springframework.web.servlet.view.document.AbstractExcelView;
11.   import com.neo.spring.vo.Account;
12.
13.   public class AccountDetailsExcelView extends AbstractExcelView {
14.         @Override
15.         protected void buildExcelDocument(Map<String, Object> model,
16.                     HSSFWorkbook workbook, HttpServletRequest request,
17.                     HttpServletResponse response) throws Exception {
18.
19.               HSSFSheet sheet = workbook.createSheet("Account Details");
20.               HSSFRow headerRow = sheet.createRow(0);
21.               headerRow.createCell((short) 0).setCellValue("ACCNO");
22.               headerRow.createCell((short) 1).setCellValue("NAME");
23.               headerRow.createCell((short) 2).setCellValue("BALANCE");
24.
25.               List<Account> accounts = (List<Account>)   .
26.                                                   model.get("accounts");
27.               if (accounts == null || accounts.isEmpty()) {
28.
29.                     sheet.createRow(1).createCell((short)1).setCellValue(
30.                                                "NO ACCOUNT DETAILS FOUND");
31.                     return;
32.               }
33.               int rowNum=1;
34.               for(Account acc:accounts)
35.               {
36.                     HSSFRow row = sheet.createRow(rowNum);
37.
38.                    row.createCell((short)0).setCellValue(acc.getAccno());
39.                     row.createCell((short)1).setCellValue(acc.getName());
40.                  row.createCell((short)2).setCellValue(acc.getBalance());
41.                     rowNum++;
42.               }
43.
44.         }
45.   }
```

spring-beans.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xmlns:p="http://www.springframework.org/schema/p"
```

```
5.    xsi:schemaLocation="http://www.springframework.org/schema/beans
6.    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7.
8.    <!-- Services configuration -->
9.
10.       <bean id="as" class="com.neo.spring.service.AccountService" />
11.
12.       <!-- Services configuration -->
13.
14.       <!-- Controllers configuration -->
15.       <bean id="wc"
16.             class="com.neo.spring.controller.AccountController">
17.             <property name="service" ref="as" />
18.       </bean>
19.       <!-- Controllers configuration -->
20.
21.       <!-- Handler Mappings configuration-->
22.       <bean id="suhm" class="org.springframework.web.servlet.handler
23.                                              .SimpleUrlHandlerMapping">
24.             <property name="mappings">
25.                   <props>
26.                         <prop key="/accountsExcel.sekhar">wc</prop>
27.                   </props>
28.             </property>
29.       </bean>
30.       <!-- Handler Mappings configuration -->
31.
32.       <!-- View resolver configurations -->
33.
34.     <bean class="org.springframework.web.servlet.view.XmlViewResolver">
35.             <property name="location">
36.                   <value>/WEB-INF/spring-excel-views.xml</value>
37.             </property>
38.       </bean>
39.
40.       <!-- View resolver configurations -->
41.
42.   </beans>
```

spring-excel-views.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:p="http://www.springframework.org/schema/p"
5.    xsi:schemaLocation="http://www.springframework.org/schema/beans
6. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7.
8.    <bean id="accountDetailsExcelView"
9.                  class="com.neo.spring.view.AccountDetailsExcelView" />
10.
11.   </beans>
```

web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
```

```
3.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5.    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
6.
7.    <servlet>
8.          <servlet-name>front-controller</servlet-name>
9.          <servlet-class>
10.                org.springframework.web.servlet.DispatcherServlet
11.         </servlet-class>
12.               <init-param>
13.                     <param-name>contextConfigLocation</param-name>
14.                     <param-value>/WEB-INF/spring-beans.xml</param-value>
15.               </init-param>
16.         </servlet>
17.         <servlet-mapping>
18.               <servlet-name>front-controller</servlet-name>
19.               <url-pattern>*.sekhar</url-pattern>
20.         </servlet-mapping>
21.         <welcome-file-list>
22.               <welcome-file>index.jsp</welcome-file>
23.         </welcome-file-list>
24.   </web-app>
```

index.jsp

```
1. <html>
2.   <body bgcolor="pink" >
3.     <center>
4.          <h1>XmlViewReslover demo application</h1>
5.          <hr>
6.          <h3> <a  href="./accountsExcel.sekhar" >Get Account
7.                                                  details</a></h3>
8.     </center>
9.     </body>
10.  </html>
```

AbstractJExcelView

Spring MVC comes with **AbstractJExcelView** class to export data to Excel file via **JExcelAPI** library. This example shows the use of **AbstractJExcelView** class in Spring MVC application to export data to Excel file for download.

1. JExcelAPI

Get the JExcelAPI library.

2. Controller

A controller class, generate dummy data for demonstration, and get the request parameter to determine which view to return. If the request parameter is equal to "EXCEL", then return an Excel view (**AbstractJExcelView**).

AccountController.java

```
//...

public class AccountController extends AbstractController{
```

```
        @Override
        protected ModelAndView handleRequestInternal(HttpServletRequest
                    request,HttpServletResponse response) throws Exception {

       // Here we are calling the method on service class
              return new ModelAndView("accountDetailsExcelView", "accounts",
                                                                  accounts);
        }
}
```

3. AbstractJExcelView

Create an Excel view by extends the **AbstractJExcelView** class, and override the **buildExcelDocument()** method to populate the data to Excel file. The **AbstractJExcelView** is using the **JExcelAPI** to create the Excel file detail.

AccountDetailsExcelView.java

```
//...
public class AccountDetailsJExcelView extends AbstractJExcelView{

        @Override
        protected void buildExcelDocument(Map<String, Object> model,
                        WritableWorkbook workbook, HttpServletRequest request,
                        HttpServletResponse response) throws Exception {
 //Here we get the account details from model and checking the condition and
 //Displaying the Account details.

        }
}
```

4. Spring Configuration

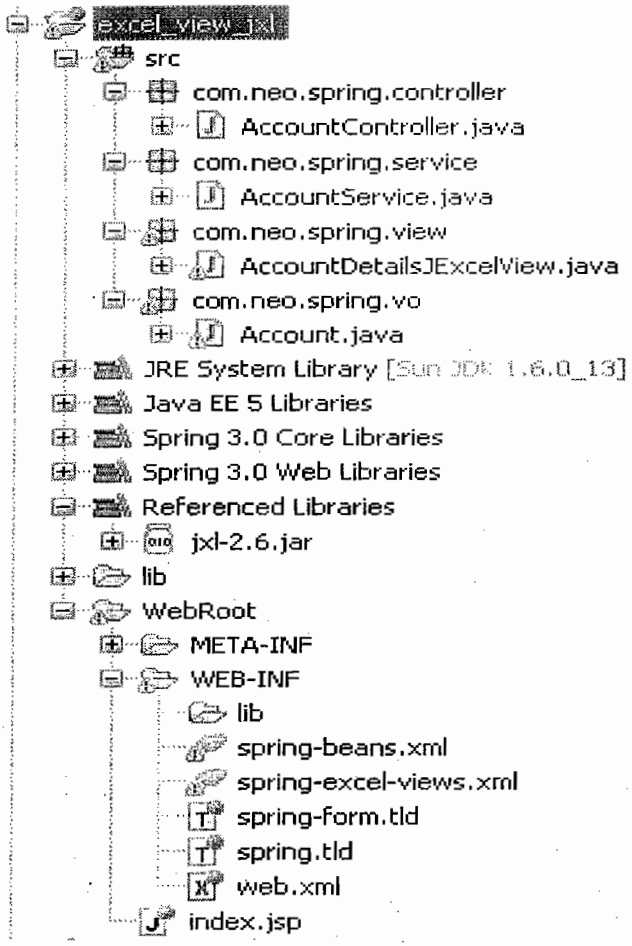
Create a **XmlViewResolver** for the Excel view.

```
<beans ...>
                <!-- Handler  Configuration -->

                <!-- Controller Configuration -->

    <bean class="org.springframework.web.servlet.view.XmlViewResolver">
       <property name="location" value="/WEB-INF/spring-excel-views.xml" />
    </bean>

</beans>
```

spring-excel-views.xml

```
<beans ...">

   <bean id="accountDetailsExcelView"
         class="com.neo.spring.view.AccountDetailsJExcelView">
   </bean>
</beans>
```

Now we will see complete application:



Account.java

```
1. package com.neo.spring.vo;
2. public class  Account
3. {
4.     private int accno;
5.     private String name;
6.     private float balance;
7.     public void setAccno(int accno)
8.     {
9.           this.accno=accno;
10.    }
11.    public  int getAccno()
12.    {
13.          return accno;
14.    }
15.    public  void setName(String name)
16.    {
17.          this.name=name;
18.    }
19.    public  String getName()
20.    {
21.          return name;
22.    }
23.    public void setBalance(float balance)
24.    {
```

```
25.         this.balance=balance;
26.    }
27.    public float getBalance()
28.    {
29.         return balance;
30.    }
31.    }
```

AccountController.java

```
1. package com.neo.spring.controller;
2.
3. import java.util.List;
4. import javax.servlet.http.HttpServletRequest;
5. import javax.servlet.http.HttpServletResponse;
6. import org.springframework.web.servlet.ModelAndView;
7. import org.springframework.web.servlet.mvc.AbstractController;
8. import com.neo.spring.service.AccountService;
9. import com.neo.spring.vo.Account;
10.
11.   public class AccountController extends AbstractController {
12.         private AccountService service;
13.
14.         public void setService(AccountService service) {
15.               this.service = service;
16.         }
17.
18.         @Override
19.         protected ModelAndView handleRequestInternal(HttpServletRequest
20.                   request, HttpServletResponse response) throws Exception {
21.               List<Account> accounts = service.getAccounts();
22.               return new ModelAndView("accountDetailsExcelView",
23.                                                   "accounts", accounts);
24.         }
25.   }
```

AccountService.java

```
1. package com.neo.spring.service;
2.
3. import java.util.ArrayList;
4. import java.util.List;
5. import com.neo.spring.vo.Account;
6.
7. public class AccountService {
8.    public List<Account> getAccounts() {
9.         List<Account> accounts = new ArrayList<Account>();
10.              Account account = null;
11.              for (int i = 0; i < 10; i++) {
12.                   account = new Account();
13.                   account.setAccno(i);
14.                   account.setName("Sekhar-" + i);
15.                   account.setBalance(i * 10);
16.                   accounts.add(account);
17.              }
18.              return accounts;
19.         }
```

```
20.   }
```

AccountDetailsPdfView.java

```
1. package com.neo.spring.view;
2.
3. import java.util.List;
4. import java.util.Map;
5. import javax.servlet.http.HttpServletRequest;
6. import javax.servlet.http.HttpServletResponse;
7. import jxl.write.Label;
8. import jxl.write.WritableSheet;
9. import jxl.write.WritableWorkbook;
10.   import
11.        org.springframework.web.servlet.view.document.AbstractJExcelView;
12.   import com.neo.spring.vo.Account;
13.
14.   public class AccountDetailsJExcelView extends AbstractJExcelView {
15.         @Override
16.         protected void buildExcelDocument(Map<String, Object> model,
17.                     WritableWorkbook workbook, HttpServletRequest
18.          request, HttpServletResponse response) throws Exception {
19.
20.         WritableSheet sheet = workbook.createSheet("Account Details", 0);
21.               sheet.addCell(new Label(0, 0, "ACCNO"));
22.               sheet.addCell(new Label(1, 0, "NAME"));
23.               sheet.addCell(new Label(2, 0, "BALANCE"));
24.
25.               List<Account> accounts = (List<Account>)
26.                                                   model.get("accounts");
27.               if (accounts == null || accounts.isEmpty()) {
28.                     sheet.addCell(new Label(0, 1, "NO ACCOUNT DETAILS
29.                                                               FOUND"));
30.                     return;
31.               }
32.               int rowNum = 1;
33.               for (Account acc : accounts) {
34.                 sheet.addCell(new Label(0, rowNum, acc.getAccno() + ""));
35.                 sheet.addCell(new Label(1, rowNum, acc.getName()));
36.               sheet.addCell(new Label(2, rowNum, acc.getBalance() + ""));
37.                     rowNum++;
38.               }
39.         }
40.   }
```

spring-beans.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xmlns:p="http://www.springframework.org/schema/p"
5.     xsi:schemaLocation="http://www.springframework.org/schema/beans
6.     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7.
8.     <!-- Services configuration -->
9.
10.         <bean id="as" class="com.neo.spring.service.AccountService" />
```

```
11.
12.         <!-- Services configuration -->
13.
14.         <!-- Controllers configuration -->
15.         <bean id="wc"
16.                   class="com.neo.spring.controller.AccountController">
17.               <property name="service" ref="as" />
18.         </bean>
19.         <!-- Controllers configuration -->
20.
21.         <!-- Handler Mappings configuration-->
22.         <bean id="suhm" class="org.springframework.web.servlet.handler
23.                                              .SimpleUrlHandlerMapping">
24.               <property name="mappings">
25.                     <props>
26.                           <prop key="/accountsExcel.sekhar">wc</prop>
27.                     </props>
28.               </property>
29.         </bean>
30.         <!-- Handler Mappings configuration -->
31.
32.         <!-- View resolver configurations -->
33.
34.   <bean class="org.springframework.web.servlet.view.XmlViewResolver">
35.               <property name="location">
36.                     <value>/WEB-INF/spring-excel-views.xml</value>
37.               </property>
38.         </bean>
39.
40.         <!-- View resolver configurations -->
41.
42.   </beans>
```

spring-pdf-views.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xmlns:p="http://www.springframework.org/schema/p"
5.     xsi:schemaLocation="http://www.springframework.org/schema/beans
6.     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7.
8.     <bean id="accountDetailsExcelView"
9.           class="com.neo.spring.view.AccountDetailsJExcelView" />
10.
11.   </beans>
```

web.xml

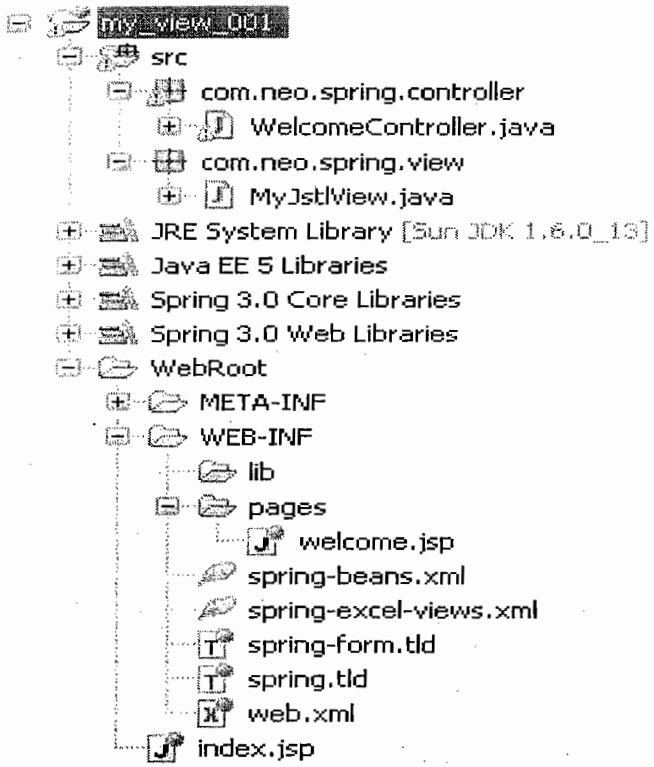
```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5.     http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
6.
7.     <servlet>
8.           <servlet-name>front-controller</servlet-name>
```

```
9.          <servlet-class>
10.                   org.springframework.web.servlet.DispatcherServlet
11.               </servlet-class>
12.               <init-param>
13.                    <param-name>contextConfigLocation</param-name>
14.                    <param-value>/WEB-INF/spring-beans.xml</param-value>
15.               </init-param>
16.         </servlet>
17.         <servlet-mapping>
18.               <servlet-name>front-controller</servlet-name>
19.               <url-pattern>*.sekhar</url-pattern>
20.         </servlet-mapping>
21.         <welcome-file-list>
22.               <welcome-file>index.jsp</welcome-file>
23.         </welcome-file-list>
24.   </web-app>
```

index.jsp

```
1. <html>
2.   <body bgcolor="pink" >
3.     <center>
4.          <h1>XmlViewReslover demo application</h1>
5.          <hr>
6.          <h3> <a  href="./accountsExcel.sekhar" >Get Account
7.                                                  details</a></h3>
8.     </center>
9.   </body>
10.   </html>
```

If we want to present new view technology which is not there in Spring framework then we have an interface View. By implementing View interface and overriding getContentType() and render() methods.



WelcomeController.java

```
1. package com.neo.spring.controller;
2.
3. import java.util.HashMap;
4. import java.util.Map;
5. import javax.servlet.http.HttpServletRequest;
6. import javax.servlet.http.HttpServletResponse;
7. import org.springframework.web.servlet.ModelAndView;
8. import org.springframework.web.servlet.mvc.AbstractController;
9.
10.   public class WelcomeController extends AbstractController {
11.         @Override
12.         protected ModelAndView handleRequestInternal(HttpServletRequest
13.                         request, HttpServletResponse response) throws Exception {
14.               String name = request.getParameter("name");
15.               String welcomeMessage = null;
16.               if (name != null && name.trim().length() > 0) {
17.                     welcomeMessage = "Welcome " + name;
18.               } else {
19.                     welcomeMessage = "Welcome Guest";
20.               }
21.               ModelAndView modelAndView = new ModelAndView("welcomePage");
22.               modelAndView.addObject("welcomeMsg", welcomeMessage);
23.               modelAndView.addObject("test", "It is testing");
24.               /*
25.                * Map<String, Object> modelMap = new HashMap<String, Object>();
26.                * modelMap.put("welcomeMsg", welcomeMessage);
27.                * modelMap.put("test",
28.                * "testing");
29.                *
```

```
30.                * modelAndView.addAllObjects(modelMap);
31.                *
32.                * return modelAndView;
33.                */
34.               return modelAndView;
35.         }
36.   }
```

MyJstlView.java

```
1. package com.neo.spring.view;
2.
3. import java.util.Iterator;
4. import java.util.Map;
5. import java.util.Set;
6. import javax.servlet.http.HttpServletRequest;
7. import javax.servlet.http.HttpServletResponse;
8. import org.springframework.web.servlet.View;
9.
10.   public class MyJstlView implements View {
11.         private String url;
12.
13.         public void setUrl(String url) {
14.               this.url = url;
15.         }
16.
17.         public String getContentType() {
18.               return "text/html";
19.         }
20.
21.         public void render(Map<String, ?> modelMap, HttpServletRequest request,
22.                     HttpServletResponse response) throws Exception {
23.               Set<String> modelNames = modelMap.keySet();
24.               Iterator<String> iterator = modelNames.iterator();
25.               while (iterator.hasNext()) {
26.                     String modelName = iterator.next();
27.                     Object modelValue = modelMap.get(modelName);
28.                     request.setAttribute(modelName, modelValue);
29.               }
30.               request.getRequestDispatcher(url).forward(request, response);
31.         }
32.   }
```

spring-beans.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.        xmlns:p="http://www.springframework.org/schema/p"
5.     xsi:schemaLocation="http://www.springframework.org/schema/beans
6. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7.
8.     <!-- Controllers configuration -->
9.     <bean id="wc" class="com.neo.spring.controller.WelcomeController" />
10.         <!-- Controllers configuration -->
11.
12.         <!-- Handler Mappings configuration-->
```

```
13.         <bean id="suhm" class="org.springframework.web.servlet.handler
14.                                                     .SimpleUrlHandlerMapping">
15.               <property name="mappings">
16.                     <props>
17.                           <prop key="/welcome.sekhar">wc</prop>
18.                     </props>
19.               </property>
20.         </bean>
21.         <!-- Handler Mappings configuration -->
22.
23.         <!-- View resolver configurations -->
24.
25.         <bean class="org.springframework.web.servlet.view.XmlViewResolver">
26.               <property name="location">
27.                     <value>/WEB-INF/spring-excel-views.xml</value>
28.               </property>
29.         </bean>
30.
31.         <!-- View resolver configurations -->
32.
33.   </beans>
```

spring-excel-views.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:p="http://www.springframework.org/schema/p"
5.    xsi:schemaLocation="http://www.springframework.org/schema/beans
6. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7.
8.    <!-- Controllers configuration -->
9.    <bean id="wc" class="com.neo.spring.controller.WelcomeController" />
10.         <!-- Controllers configuration -->
11.
12.         <!-- Handler Mappings configuration-->
13.         <bean id="suhm" class="org.springframework.web.servlet.handler
14.                                                     .SimpleUrlHandlerMapping">
15.               <property name="mappings">
16.                     <props>
17.                           <prop key="/welcome.sekhar">wc</prop>
18.                     </props>
19.               </property>
20.         </bean>
21.         <!-- Handler Mappings configuration -->
22.
23.         <!-- View resolver configurations -->
24.
25.         <bean class="org.springframework.web.servlet.view.XmlViewResolver">
26.               <property name="location">
27.                     <value>/WEB-INF/spring-excel-views.xml</value>
28.               </property>
29.         </bean>
30.
31.         <!-- View resolver configurations -->
```

```
32.
33.   </beans>
```

web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5.     http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
6.
7.     <servlet>
8.           <servlet-name>front-controller</servlet-name>
9.     <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
10.              <init-param>
11.                    <param-name>contextConfigLocation</param-name>
12.                    <param-value>/WEB-INF/spring-beans.xml</param-value>
13.              </init-param>
14.         </servlet>
15.         <servlet-mapping>
16.              <servlet-name>front-controller</servlet-name>
17.              <url-pattern>*.sekhar</url-pattern>
18.         </servlet-mapping>
19.         <welcome-file-list>
20.              <welcome-file>index.jsp</welcome-file>
21.         </welcome-file-list>
22.   </web-app>
```

index.jsp

```
1. <html>
2.   <body bgcolor="pink" >
3.     <center>
4.          <h1>My View demo application</h1>
5.          <hr>
6.          <form action="./welcome.sekhar" method="post" >
7.               Name : <input type="text" name="name"  > <br/>
8.               <input type="submit" value="Get Welcome" >
9.          </form>
10.       </center>
11.     </body>
12.   </html>
```

welcome.jsp

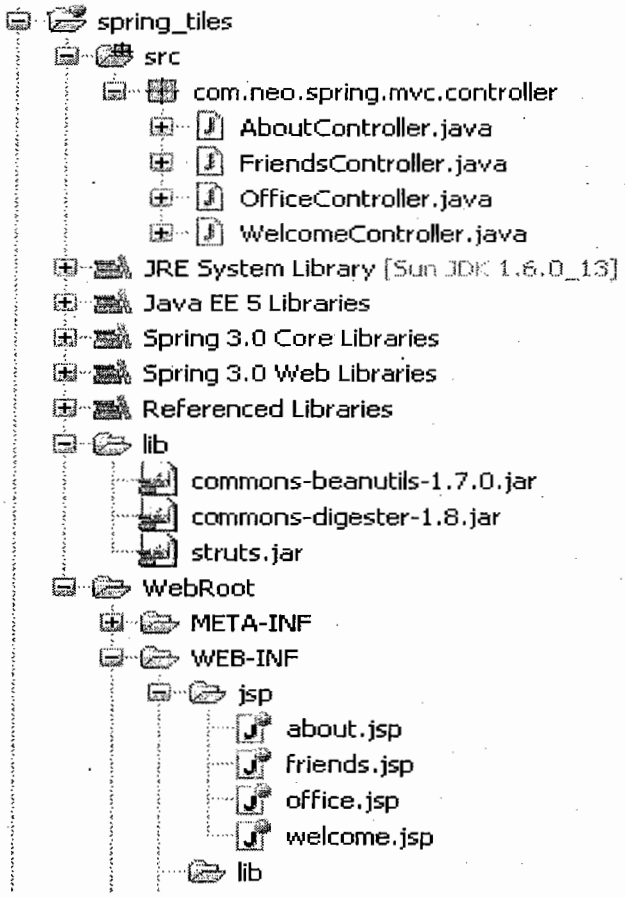
```
1. <%@page isErrorPage="true" %>
2. <html>
3.   <body bgcolor="pink" >
4. <center>
5.     <h1>My View demo application</h1>
6.     <hr/>
7.     <h2>${welcomeMsg}</h2>
8.     <h2>${test}</h2>
9. </center>
10.     </body>
11.   </html>
```

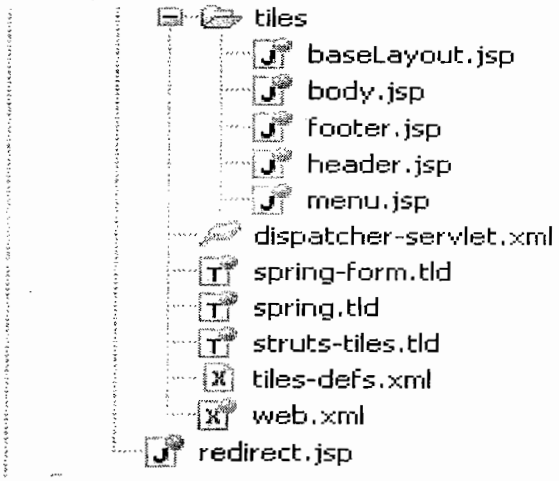
TILES

Steps to develop Tiles:

1. Spring is not given any support to develop Tiles. So, To develop Tiles we have to use Struts given jar files. We have to use three struts jar files to develop Tiles.
   - Commons-beanutils.jar
   - Commons-digester.jar
   - Struts.jar
2. We have to develop our controllers and have to configure them in spring configuration file.
3. We have to develop one layout and content jsp's.
4. In tiles-def.xml, root tag is `<tiles-definitions>` and sub tag is `<definition>`. We will specify how many tiles we want that many `<definition>` tags will be there.
5. In spring configuration file, We have to inject ***org.springframework.web.servlet.view.tiles.TilesView*** to ViewResolver.
6. We have to configure ***org.springframework.web.servlet.view.tiles.TilesConfigurer*** and ***org.apache.struts.tiles.xmlDefinition.I18nFactorySet***.
7. We have to inject I18NFactorySet and tiles-def.xml to TilesCofigurer.

Tiles-defs.xml will read by I18NFactorySet, it gives to TilesConfigurer then TilesView will contacts TilesConfigurer.





AboutController.java

```
1. package com.neo.spring.mvc.controller;
2.
3. import javax.servlet.http.HttpServletRequest;
4. import javax.servlet.http.HttpServletResponse;
5. import org.springframework.web.servlet.ModelAndView;
6. import org.springframework.web.servlet.mvc.Controller;
7.
8. public class AboutController implements Controller {
9.     public ModelAndView handleRequest(HttpServletRequest request,
10.                     HttpServletResponse response) throws Exception {
11.                 System.out.println("about is called");
12.                 return new ModelAndView("about");
13.         }
14.   }
```

FriendsController.java

```
1. package com.neo.spring.mvc.controller;
2.
3. import javax.servlet.http.HttpServletRequest;
4. import javax.servlet.http.HttpServletResponse;
5. import org.springframework.web.servlet.ModelAndView;
6. import org.springframework.web.servlet.mvc.Controller;
7.
8. public class FriendsController implements Controller {
9.
10.         public ModelAndView handleRequest(HttpServletRequest request,
11.                     HttpServletResponse response) throws Exception {
12.                 System.out.println("friends is called ");
13.                 return new ModelAndView("friends");
14.         }
15.   }
```

OfficeController.java

```
1. package com.neo.spring.mvc.controller;
2.
3. import javax.servlet.http.HttpServletRequest;
4. import javax.servlet.http.HttpServletResponse;
5. import org.springframework.web.servlet.ModelAndView;
```

```
6. import org.springframework.web.servlet.mvc.Controller;
7.
8. public class OfficeController implements Controller {
9.
10.         public ModelAndView handleRequest(HttpServletRequest request,
11.                     HttpServletResponse response) throws Exception {
12.               System.out.println("office is called");
13.               return new ModelAndView("office");
14.         }
15.   }
```

WelcomeController.java

```
1. package com.neo.spring.mvc.controller;
2.
3. import javax.servlet.http.HttpServletRequest;
4. import javax.servlet.http.HttpServletResponse;
5. import org.springframework.web.servlet.ModelAndView;
6. import org.springframework.web.servlet.mvc.Controller;
7.
8. public class WelcomeController implements Controller {
9.    public ModelAndView handleRequest(HttpServletRequest request,
10.                    HttpServletResponse response) throws Exception {
11.               System.out.println("welcome is called");
12.               return new ModelAndView("welcome");
13.         }
14.   }
```

dispatcher-servlet.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.    xmlns:p="http://www.springframework.org/schema/p"
5.    xmlns:context="http://www.springframework.org/schema/context"
6.    xsi:schemaLocation="http://www.springframework.org/schema/beans
7.    http://www.springframework.org/schema/beans/spring-beans.xsd
8.    http://www.springframework.org/schema/context
9.    http://www.springframework.org/schema/context/spring-context.xsd">
10.
11.         <!--  Controller beans configuration -->
12.         <bean id="ac"
13.               class="com.neo.spring.mvc.controller.AboutController" />
14.         <bean id="fc"
15.               class="com.neo.spring.mvc.controller.FriendsController" />
16.         <bean id="oc"
17.               class="com.neo.spring.mvc.controller.OfficeController" />
18.         <bean id="wc"
19.               class="com.neo.spring.mvc.controller.WelcomeController" />
20.
21.         <!--  Handler Mappings configuration -->
22.         <bean class="org.springframework.web.servlet.handler
23.                                          .SimpleUrlHandlerMapping">
24.               <property name="mappings">
25.                     <props>
26.                           <prop key="/about.htm">ac</prop>
27.                           <prop key="/friends.htm">fc</prop>
```

```
                        <prop key="/office.htm">oc</prop>
                        <prop key="/welcome.htm">wc</prop>
                    </props>
                </property>
        </bean>

        <!--  View Resolver configuration -->
        <bean id="viewResolver" class="org.springframework.web.servlet
                                    .view.InternalResourceViewResolver">
                <property name="viewClass"
        value="org.springframework.web.servlet.view.tiles.TilesView" />
        </bean>

        <!--  Tiles Configuration -->
        <bean id="tilesConfigurer" class="org.springframework.web.servlet
                                        .view.tiles.TilesConfigurer">
                <property name="factoryClass"
        value="org.apache.struts.tiles.xmlDefinition.I18nFactorySet" />
                <property name="definitions" value="/WEB-INF/tiles-
                                                    defs.xml" />
        </bean>

    </beans>
```

tiles-defs.xml

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

<!DOCTYPE tiles-definitions PUBLIC
"-//Apache Software Foundation//DTD Tiles Configuration 1.1//EN"
"http://jakarta.apache.org/struts/dtds/tiles-config_1_1.dtd">

<tiles-definitions>

    <definition name="baseLayout" page="/WEB-INF/tiles/baseLayout.jsp">
                <put name="title" value="Template" />
                <put name="header" value="/WEB-INF/tiles/header.jsp" />
                <put name="menu" value="/WEB-INF/tiles/menu.jsp" />
                <put name="body" value="/WEB-INF/tiles/body.jsp" />
                <put name="footer" value="/WEB-INF/tiles/footer.jsp" />
        </definition>

        <definition name="welcome" extends="baseLayout">
                <put name="title" value="Welcome" />
                <put name="body" value="/WEB-INF/jsp/welcome.jsp" />
        </definition>

        <definition name="friends" extends="baseLayout">
                <put name="title" value="Friends" />
                <put name="body" value="/WEB-INF/jsp/friends.jsp" />
        </definition>

        <definition name="office" extends="baseLayout">
                <put name="title" value="Office" />
                <put name="body" value="/WEB-INF/jsp/office.jsp" />
        </definition>
```

```
31.
32.         <definition name="about" extends="baseLayout">
33.               <put name="title" value="About Authoer"_/>
34.               <put name="body" value="/WEB-INF/jsp/about.jsp" />
35.         </definition>
36.
37.   </tiles-definitions>
```

web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xmlns="http://java.sun.com/xml/ns/javaee"
4. xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
5. xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
6. http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID"
7. version="2.5">
8.   <display-name>SpringExample16</display-name>
9.   <servlet>
10.          <servlet-name>dispatcher</servlet-name>
11.          <servlet-class>
12.                org.springframework.web.servlet.DispatcherServlet
13.          </servlet-class>
14.          <load-on-startup>1</load-on-startup>
15.      </servlet>
16.      <servlet-mapping>
17.          <servlet-name>dispatcher</servlet-name>
18.          <url-pattern>*.htm</url-pattern>
19.      </servlet-mapping>
20.      <welcome-file-list>
21.          <welcome-file>redirect.jsp</welcome-file>
22.      </welcome-file-list>
23.  </web-app>
```

about.jsp

```
1. This Application done by <br/>
2. <h1>Somasekhar Reddy. Yerragudi</h1>
```

friends.jsp

```
1. <p>More details about the Friends TV show goes here...</p>
```

office.jsp

```
1. <p>More details about the Office TV show goes here...</p>
```

welcome.jsp

```
1. <p>Welcome Guest.</p>
```

baseLayout.jsp

```
1. <%@ taglib uri="http://struts.apache.org/tags-tiles" prefix="tiles" %>
2. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
3.    "http://www.w3.org/TR/html4/loose.dtd">
4.
5. <html>
6.     <head>
7.        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
8.           <title><tiles:getAsString name="title" ignore="true" /></title>
```

```
9.      </head>
10.        <body bgcolor="wheat" >
11.            <table border="1" cellpadding="2" cellspacing="2"
12.                                                        align="center">
13.                <tr>
14.                    <td height="30" colspan="2">
15.                         <tiles:insert attribute="header" />
16.                    </td>
17.                </tr>
18.                <tr>
19.                    <td height="250">
20.                        <tiles:insert attribute="menu" />
21.                    </td>
22.                    <td width="350">
23.                        <tiles:insert attribute="body" />
24.                    </td>
25.                </tr>
26.                <tr>
27.                    <td height="30" colspan="2">
28.
29.                        <tiles:insert attribute="footer" />
30.                    </td>
31.                </tr>
32.            </table>
33.        </body>
34.    </html>
```

body.jsp

```
1. <p> sample body content.</p>
```

footer.jsp

```
1. <div align="center">&copy; sekkharit.com</div>
```

header.jsp

```
1. <div align="center" style="font-weight:bold">TV shows</div>
```

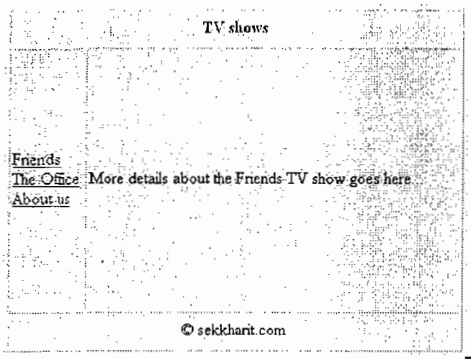
menu.jsp

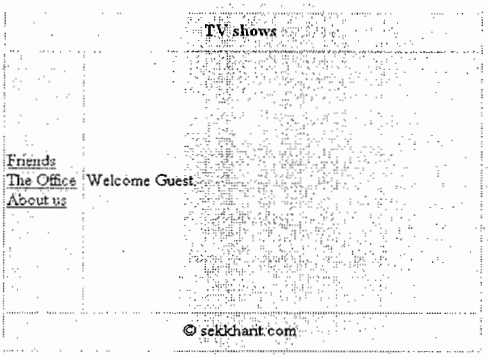
```
1. <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
2.
3. <a href="<c:url value="friends.htm"/>" >Friends</a><br>
4. <a href="<c:url value="office.htm"/>" >The Office</a><br>
5. <a href="<c:url value="about.htm"/>" >About us</a>
```

redirect.jsp

```
1. <%@page contentType="text/html" pageEncoding="UTF-8"%>
2. <% response.sendRedirect("welcome.htm"); %>
```

http://localhost:8080/spring_tiles/welcome.htm





Controllers

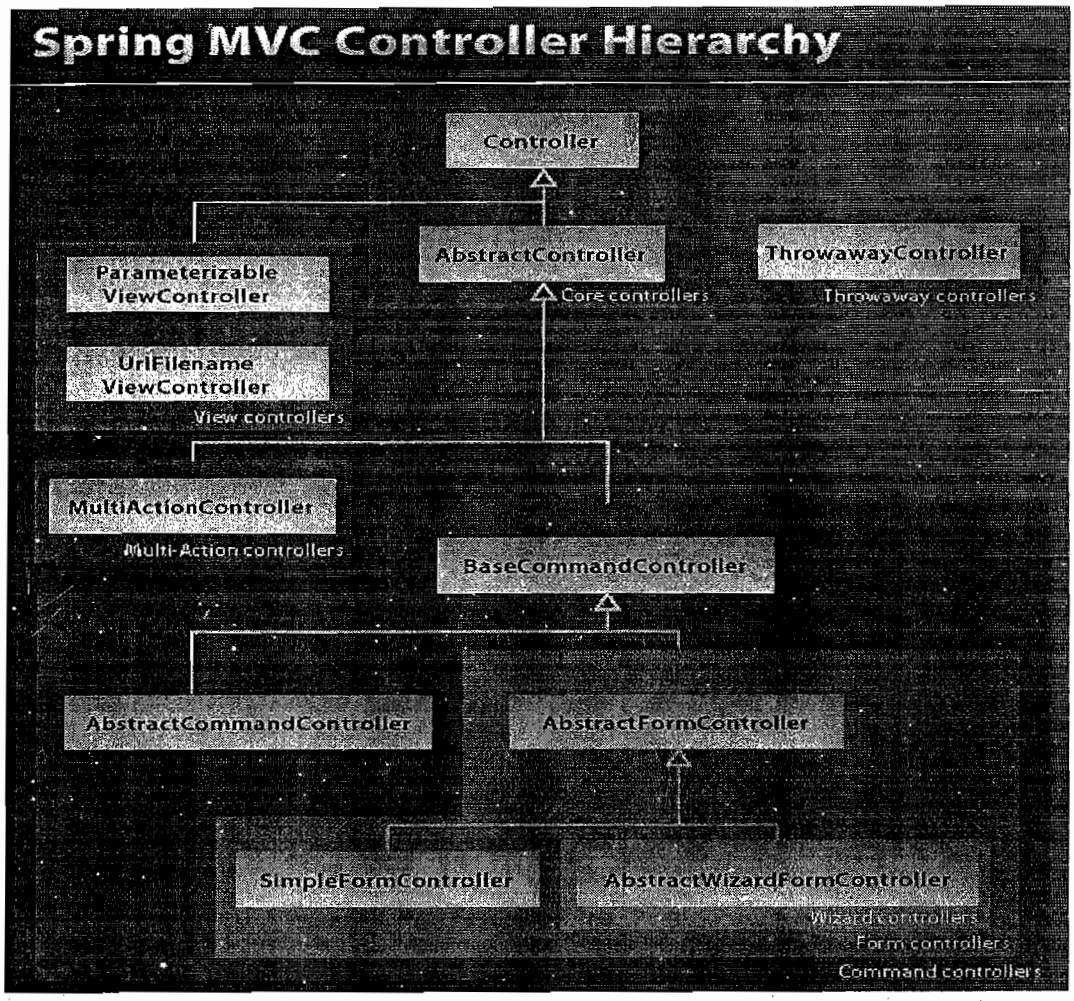
In this Spring MVC, DispatcherServlet works as the controller and it delegates the request to the Controller. Developers extends the abstract controller provided by the framework and writes the business logic there. The actual business related processing is done in the Controller.

Spring MVC provides many abstract controllers, which is designed for specific tasks. Here is the list of abstract controllers that comes with the Spring MVC module:

1. SimpleFormController
2. AbstractController
3. AbstractCommandController
4. CancellableFormController
5. AbstractCommandController
6. MultiActionController
7. ParameterizableViewController
8. ServletForwardingController
9. ServletWrappingController
10. UrlFilenameViewController

Following diagram shows the Controllers hierarchy in Spring MVC:

Spring MVC Controllers Hierarchy:



SimpleFormController

Asking the User to fill in a Form containing various information and submitting the form normally happens in almost every Web Application. The ***Simple Form Controller*** is exactly used for that purpose.

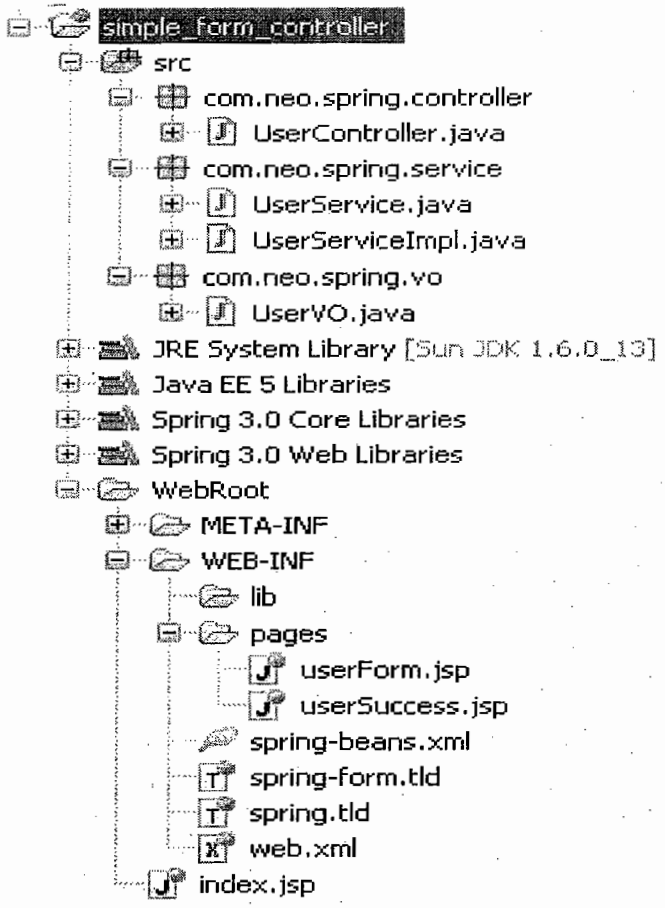
To handle forms in Spring we need to extend our controller class from SimpleFormController class The SimpleFormController is deprecated as of Spring 3.0 so if we are using Spring 3.0 or above use the annotate controllers instead.

If we are using Spring 3.0 so we see the ***SuppressWarnings*** annotation there. Here we extend the ***UserController*** from ***SimpleFormController***, this makes the controller class capable of handling forms. Usually a form will be associated with a particular domain object, in our case it is the ***UserVo*** class.

In Spring this domain object is called command object by default. To refer the command object in the jsp page we need to set the command class using the ***setCommandClass()*** method in the constructor. Let say the User class has a name property, and to refer this in the jsp page we will use "***command.name***". We can also change this name by using the ***setCommandName()*** method. Here we set the name to user, so to access the user name in the jsp page we use "*user.name*".

We need to have a method to handle the form when the form is submitted, here ***onSubmit()*** method is used for this purpose. The ***onSubmit()*** method has access to the command object, we first typecast the command object to *UserVo* (our domain object) and then to register the user we call the ***register()*** method of the service class and finally return the ***ModelandView*** object.

All the forms field values will be submitted as Strings to the form controller. Spring has several pre registered property editors to convert the String values to common data types. Incase we have a custom data type we need to create custom property editors to handle them.
Let us see an simple example on this.



UserController.java

```
1. package com.neo.spring.controller;
2.
3. import org.springframework.web.servlet.ModelAndView;
4. import org.springframework.web.servlet.mvc.SimpleFormController;
5. import com.neo.spring.service.UserService;
6. import com.neo.spring.vo.UserVO;
7.
8. @SuppressWarnings("deprecation")
9. public class UserController extends SimpleFormController {
10.
11.         private UserService userService;
12.
13.         public void setUserService(UserService userService) {
14.               this.userService = userService;
15.         }
16.
17.         public UserController() {
18.               setCommandClass(UserVO.class);
19.               setCommandName("user");
20.         }
21.
22.         @Override
23.         protected ModelAndView onSubmit(Object command)throws Exception {
24.               UserVO userVO = (UserVO) command;
```

```
25.                 userService.register(userVO);
26.                 return new ModelAndView(getSuccessView(), "user", userVO);
27.         }
28.   }
```

UserService.java

```
1. package com.neo.spring.service;
2. import com.neo.spring.vo.UserVO;
3. public interface UserService {
4.
5.    public void register(UserVO userVO);
6. }
```

UserServiceImpl.java

```
1. package com.neo.spring.service;
2. import com.neo.spring.vo.UserVO;
3. public class UserServiceImpl implements UserService {
4.
5.    public void register(UserVO userVO) {
6.          // Persist the user object here.
7.          System.out.println("User added successfully");
8.    }
9. }
```

UserVO.java

```
1. package com.neo.spring.vo;
2. public class UserVO {
3.
4.    private String name;
5.    private String password;
6.    private String gender;
7.    private String country;
8.    private String aboutYou;
9.    private String[] community;
10.         private Boolean mailingList;
11.
12.         public String getName() {
13.               return name;
14.         }
15.         public void setName(String name) {
16.               this.name = name;
17.         }
18.         public String getPassword() {
19.               return password;
20.         }
21.         public void setPassword(String password) {
22.               this.password = password;
23.         }
24.         public String getGender() {
25.               return gender;
26.         }
27.         public void setGender(String gender) {
28.               this.gender = gender;
29.         }
30.         public String getCountry() {
```

```
31.             return country;
32.         }
33.         public void setCountry(String country) {
34.             this.country = country;
35.         }
36.         public String getAboutYou() {
37.             return aboutYou;
38.         }
39.         public void setAboutYou(String aboutYou) {
40.             this.aboutYou = aboutYou;
41.         }
42.         public String[] getCommunity() {
43.             return community;
44.         }
45.         public void setCommunity(String[] community) {
46.             this.community = community;
47.         }
48.         public Boolean getMailingList() {
49.             return mailingList;
50.         }
51.         public void setMailingList(Boolean mailingList) {
52.             this.mailingList = mailingList;
53.         }
54.   }
```

spring-beans.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xmlns:p="http://www.springframework.org/schema/p"
5.     xsi:schemaLocation="http://www.springframework.org/schema/beans
6.     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7.
8.     <!-- Services configuration -->
9.     <bean id="us" class="com.neo.spring.service.UserServiceImpl" />
10.
11.         <!-- Controllers configuration -->
12.         <bean name="uc" class="com.neo.spring.controller.UserController">
13.             <property name="userService" ref="us" />
14.             <property name="formView" value="userForm" />
15.             <property name="successView" value="userSuccess" />
16.         </bean>
17.
18.         <!-- Handler Mappings configuration-->
19.         <bean id="hm" class="org.springframework.web.servlet.handler
20.                                             .SimpleUrlHandlerMapping">
21.             <property name="mappings">
22.                 <props>
23.                     <prop key="/userRegistration.sekhar">uc</prop>
24.                 </props>
25.             </property>
26.         </bean>
27.
28.         <!-- View resolver configurations -->
29.         <bean id="viewResolver" class="org.springframework.web.servlet
```

```
30.                                          .view.InternalResourceViewResolver">
31.              <property name="prefix" value="/WEB-INF/pages/" />
32.              <property name="suffix" value=".jsp" />
33.         </bean>
34.
35.   </beans>
```

web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5.     http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
6.
7.     <servlet>
8.           <servlet-name>front-controller</servlet-name>
9.           <servlet-class>
10.                org.springframework.web.servlet.DispatcherServlet
11.    </servlet-class>
12.              <init-param>
13.                   <param-name>contextConfigLocation</param-name>
14.                   <param-value>/WEB-INF/spring-beans.xml</param-value>
15.              </init-param>
16.          </servlet>
17.          <servlet-mapping>
18.               <servlet-name>front-controller</servlet-name>
19.               <url-pattern>*.sekhar</url-pattern>
20.          </servlet-mapping>
21.          <welcome-file-list>
22.               <welcome-file>index.jsp</welcome-file>
23.          </welcome-file-list>
24.    </web-app>
```

index.jsp

```
1. <%@page contentType="text/html" pageEncoding="UTF-8"%>
2. <% response.sendRedirect("userRegistration.sekhar"); %>
```

userForm.jsp

```
1. <%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
2. <html>
3. <body bgcolor="pink">
4. <center>
5.     <h1>SimpleFormController Demo application</h1>
6.     <hr/><hr/>
7.
8. <form:form  method="POST" commandName="user">
9. <table>
10.              <tr>
11.                   <td>User Name :</td>
12.                   <td><form:input path="name" /></td>
13.              </tr>
14.              <tr>
15.                   <td>Password :</td>
16.                   <td><form:password path="password" /></td>
17.              </tr>
```

```
18.                 <tr>
19.                       <td>Gender :</td>
20.                 <td><form:radiobutton path="gender" value="M" label="M" />
21.                 <form:radiobutton path="gender" value="F" label="F" /></td>
22.                 </tr>
23.                 <tr>
24.                       <td>Country :</td>
25.                       <td><form:select path="country">
26.                             <form:option value="0" label="Select" />
27.                             <form:option value="1" label="India" />
28.                             <form:option value="2" label="USA" />
29.                             <form:option value="3" label="UK" />
30.                       </form:select></td>
31.                 </tr>
32.                 <tr>
33.                       <td>About you :</td>
34.                       <td><form:textarea path="aboutYou" /></td>
35.                 </tr>
36.                 <tr>
37.                       <td>Community :</td>
38.                       <td>
39.                             <form:checkbox path="community" value="Spring"
40.                                                         label="Spring" />
41.                             <form:checkbox path="community"
42.                                         value="Hibernate" label="Hibernate" />
43.                             <form:checkbox path="community" value="Struts"
44.                                                         label="Struts" />
45.                       </td>
46.                 </tr>
47.                 <tr>
48.                       <td></td>
49.                       <td><form:checkbox path="mailingList"
50.                 label="Would you like to join our mailinglist?" /></td>
51.                 </tr>
52.                 <tr>
53.                       <td colspan="2"><input type="submit"></td>
54.                 </tr>
55.           </table>
56.     </form:form>
57.     </center>
58.     </body>
59.     </html>
```

Here the ***path*** attribute is used to bind the form fields to the domain object. Here we use the *HTTP POST* method to submit the form. Inorder to bind the form fields to the domain object successfully the command object should be set to the same name in the jsp page and the controller class. To set the command object name in the jsp page, use the ***commandName*** attribute of the form tag.

userSuccess.jsp

```
1. <html>
2. <body bgcolor="pink" >
3. <center>
4.     <h1>SimpleFormController Demo application</h1>
5.     <hr/><hr/>
6.
```

```
7.    <h2>User Details</h2>
8.    User Name    : ${user.name} <br/>
9.    Gender       : ${user.gender} <br/>
10.         Country      : ${user.country} <br/>
11.         About You    : ${user.aboutYou} <br/>
12.         Community    : ${user.community[0]}  ${user.community[1]}
13.                        ${user.community[2]}<br/>
14.         Mailing List: ${user.mailingList}
15.   </center>
16.   </body>
17.   </html>
```

When the form is submitted (during the *HTTP POST* request) the onSubmit() method of the UserController class will be called, on successful execution of the method the ***successView*** will be rendered. Incase of any type conversion errors or validation errors the ***formView*** will be automatically displayed.
The *onSubmit()* method of the *UserController* class will be called and the control will be transfered to the view "*successView*". We use *InternalResourceViewResolver* here, so the *userSuccess.jsp* page will be dispalyed. In the *userSuccess.jsp* page we dispaly all the user details using the jstl tags.

FormValidation

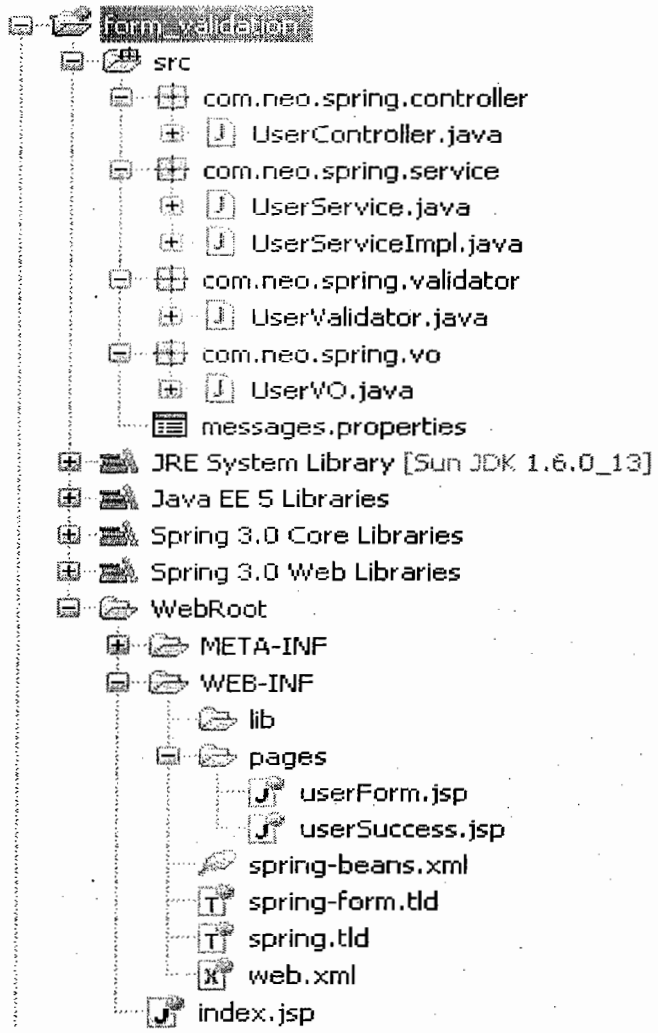
In this example we will see how to validate the user form. To validate the form fields all we need to do is to have a separate *UserValidator* class that implements the *Validator* interface, override the *validate()* method to perform all the validations.

In the jsp page add *form:errors* tag to dispaly errors and finally we need to link the *UserValidator* class with the *UserController* class. Since the *UserController* class extends the *SimpleFormController* class the *validate()* method will be called automatically.

Incase we would like to have the error messages in a seperate properties file then we need to create a new properties file, add all the error keys and its corresponding values and finally link it to Spring configuration file.

Here we need to override the validate method, to check mandatory condition we can use the *ValidationUtils* class methods like *rejectIfEmptyOrWhitespace()* or *rejectIfEmpty() methods*. These methods takes three arguments the Errors object, the property name, and the error code. Here we have the error messages in a seperate properties file so we add the error code, we can even add the error messages directly. To do any other validation we have access to the domain object, using the domain object validate the properties, incase there are errors we can use the *rejectValue()* method of the Errors class to add errors. Here the first argument is the property name and the second argument is the error code.

The *path* attribute of the *form:errors* tag indicate the property for which the error should be displayed. Use *path="*"* to display all the errors together.
In the Spring configuration file, configure the UserValidator and inject to the UserController and add the properties file.



UserController.java

```
1. package com.neo.spring.controller;
2.
3. import org.springframework.web.servlet.ModelAndView;
4. import org.springframework.web.servlet.mvc.SimpleFormController;
5. import com.neo.spring.service.UserService;
6. import com.neo.spring.vo.UserVO;
7.
8. @SuppressWarnings("deprecation")
9. public class UserController extends SimpleFormController {
10.
11.         private UserService userService;
12.
13.         public UserController() {
14.               setCommandClass(UserVO.class);
15.               setCommandName("user");
16.         }
17.
18.         public void setUserService(UserService userService) {
19.               this.userService = userService;
20.         }
21.
```

```
22.         @Override
23.         protected ModelAndView onSubmit(Object command) throws Exception {
24.               UserVO userVO = (UserVO) command;
25.               userService.register(userVO);
26.               return new ModelAndView(getSuccessView(), "user", userVO);
27.         }
28.
29.   }
```

UserService.java

```
1. package com.neo.spring.service;
2. import com.neo.spring.vo.UserVO;
3. public interface UserService {
4.
5.    public void register(UserVO userVO);
6. }
```

UserServiceImpl.java

```
1. package com.neo.spring.service;
2. import com.neo.spring.vo.UserVO;
3. public class UserServiceImpl implements UserService {
4.
5.    public void register(UserVO userVO) {
6.          // Persist the user object here.
7.          System.out.println("User added successfully");
8.
9.    }
10.   }
```

UserValidator.java

```
1. package com.neo.spring.validator;
2.
3. import org.springframework.validation.Errors;
4. import org.springframework.validation.ValidationUtils;
5. import org.springframework.validation.Validator;
6. import com.neo.spring.vo.UserVO;
7.
8. public class UserValidator implements Validator {
9.
10.         public boolean supports(Class<?> clazz) {
11.               return UserVO.class.isAssignableFrom(clazz);
12.         }
13.
14.         public void validate(Object target, Errors errors) {
15.               ValidationUtils.rejectIfEmptyOrWhitespace(errors, "name",
16.                                                         "name.required");
17.               ValidationUtils.rejectIfEmptyOrWhitespace(errors,
18.                                         "password","password.required");
19.               ValidationUtils.rejectIfEmpty(errors, "gender",
20.                                                "gender.required");
21.               ValidationUtils.rejectIfEmpty(errors, "country",
22.                                                "country.required");
23.               ValidationUtils.rejectIfEmptyOrWhitespace(errors,
24.                                       "aboutYou", "aboutYou.required");
25.               UserVO user = (UserVO) target;
```

```
26.
27.   if (user.getCommunity() == null || user.getCommunity().length == 0) {
28.               errors.rejectValue("community", "community.required");
29.               }
30.         }
31.   }
```

UserVO.java

```
1. package com.neo.spring.vo;
2.
3. public class UserVO {
4.    private String name;
5.    private String password;
6.    private String gender;
7.    private String country;
8.    private String aboutYou;
9.    private String[] community;
10.         private Boolean mailingList;
11.
12.         public String getName() {
13.               return name;
14.         }
15.         public void setName(String name) {
16.               this.name = name;
17.         }
18.         public String getPassword() {
19.               return password;
20.         }
21.         public void setPassword(String password) {
22.               this.password = password;
23.         }
24.         public String getGender() {
25.               return gender;
26.         }
27.         public void setGender(String gender) {
28.               this.gender = gender;
29.         }
30.         public String getCountry() {
31.               return country;
32.         }
33.         public void setCountry(String country) {
34.               this.country = country;
35.         }
36.         public String getAboutYou() {
37.               return aboutYou;
38.         }
39.         public void setAboutYou(String aboutYou) {
40.               this.aboutYou = aboutYou;
41.         }
42.         public String[] getCommunity() {
43.               return community;
44.         }
45.         public void setCommunity(String[] community) {
46.               this.community = community;
47.         }
```

```
48.         public Boolean getMailingList() {
49.               return mailingList;
50.         }
51.         public void setMailingList(Boolean mailingList) {
52.               this.mailingList = mailingList;
53.         }
54.   }
```

spring-beans.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xmlns:p="http://www.springframework.org/schema/p"
5.     xsi:schemaLocation="http://www.springframework.org/schema/beans
6.     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7.
8.     <!-- Services configuration -->
9.     <bean id="us" class="com.neo.spring.service.UserServiceImpl" />
10.
11.          <!-- Validators configuration -->
12.          <bean id="userValidator"
13.                class="com.neo.spring.validator.UserValidator" />
14.
15.          <!-- properties files configuration -->
16.          <bean id="messageSource" class="org.springframework.context
17.                                    .support.ResourceBundleMessageSource">
18.                <property name="basename" value="messages" />
19.          </bean>
20.
21.          <!-- Controllers configuration -->
22.          <bean name="uc" class="com.neo.spring.controller.UserController">
23.                <property name="validator" ref="userValidator" />
24.                <property name="userService" ref="us" />
25.                <property name="formView" value="userForm" />
26.                <property name="successView" value="userSuccess" />
27.          </bean>
28.
29.          <!-- Handler Mappings configuration-->
30.          <bean id="hm" class="org.springframework.web.servlet.handler
31.                                              .SimpleUrlHandlerMapping">
32.                <property name="mappings">
33.                      <props>
34.                            <prop key="/userRegistration.sekhar">uc</prop>
35.                      </props>
36.                </property>
37.          </bean>
38.
39.          <!-- View resolver configurations -->
40.          <bean id="viewResolver" class="org.springframework.web.servlet
41.                                        .view.InternalResourceViewResolver">
42.                <property name="prefix" value="/WEB-INF/pages/" />
43.                <property name="suffix" value=".jsp" />
44.          </bean>
45.
46.   </beans>
```

web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5.     http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
6.
7.     <servlet>
8.           <servlet-name>front-controller</servlet-name>
9.           <servlet-class>
10.                    org.springframework.web.servlet.DispatcherServlet
11.          </servlet-class>
12.          <init-param>
13.                <param-name>contextConfigLocation</param-name>
14.                <param-value>/WEB-INF/spring-beans.xml</param-value>
15.          </init-param>
16.    </servlet>
17.    <servlet-mapping>
18.          <servlet-name>front-controller</servlet-name>
19.          <url-pattern>*.sekhar</url-pattern>
20.    </servlet-mapping>
21.    <welcome-file-list>
22.          <welcome-file>index.jsp</welcome-file>
23.    </welcome-file-list>
24. </web-app>
```

index.jsp

```
1. <%@page contentType="text/html" pageEncoding="UTF-8"%>
2. <% response.sendRedirect("userRegistration.sekhar"); %>
```

userForm.jsp

```
1. <%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
2. <html>
3. <head>
4. <style>
5. .error {
6. color: BLUE;
7. font-style: italic;
8. }
9. </style>
10.
11.  </head>
12.  <body bgcolor="pink">
13.  <center>
14.        <h1>SimpleFormController Demo application</h1>
15.        <hr/><hr/>
16.
17.  <form:form  method="POST" commandName="user">
18.        <table>
19.              <tr>
20.                    <td>User Name :</td>
21.                    <td><form:input path="name" /></td>
22.                    <td><form:errors path="name" cssClass="error" /></td>
```

```
23.                 </tr>
24.                 <tr>
25.                       <td>Password :</td>
26.                       <td><form:password path="password" /></td>
27.                 <td><form:errors path="password" cssClass="error" /></td>
28.                 </tr>
29.                 <tr>
30.                       <td>Gender :</td>
31.                 <td><form:radiobutton path="gender" value="M" label="M" />
32.                 <form:radiobutton path="gender" value="F" label="F" /></td>
33.                 <td><form:errors path="gender" cssClass="error" /></td>
34.                 </tr>
35.                 <tr>
36.                       <td>Country :</td>
37.                       <td><form:select path="country">
38.                             <form:option value="" label="Select" />
39.                             <form:option value="1" label="India" />
40.                             <form:option value="2" label="USA" />
41.                             <form:option value="3" label="UK" />
42.                       </form:select></td>
43.                 <td><form:errors path="country" cssClass="error" /></td>
44.                 </tr>
45.                 <tr>
46.                       <td>About you :</td>
47.                       <td><form:textarea path="aboutYou" /></td>
48.                 <td><form:errors path="aboutYou" cssClass="error" /></td>
49.                 </tr>
50.                 <tr>
51.                       <td>Community :</td>
52.                       <td><form:checkbox path="community" value="Spring"
53.                             label="Spring" />
54.                       <form:checkbox path="community" value="Hibernate"
55.                             label="Hibernate" />
56.                       <form:checkbox path="community" value="Struts"
57.                             label="Struts" /></td>
58.                 <td><form:errors path="community" cssClass="error" /></td>
59.                 </tr>
60.                 <tr>
61.                       <td colspan="3"><form:checkbox path="mailingList"
62.                 label="Would you like to join our mailinglist?" /></td>
63.                 </tr>
64.                 <tr>
65.                 <td colspan="3"><input type="submit" value="Register"></td>
66.                 </tr>
67.           </table>
68.
69.     </form:form>
70.     </center>
71.     </body>
72.     </html>
```

userSuccess.jsp

```
1. <html>
2. <body bgcolor="pink" >
3. <center>
```

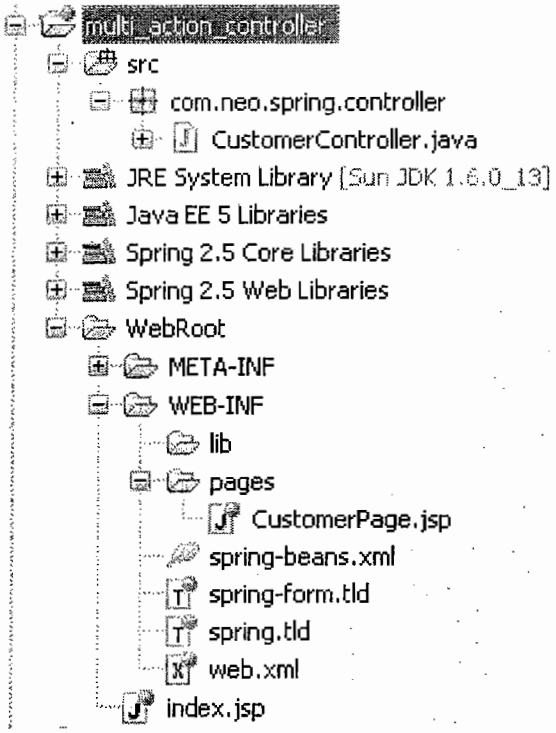
```
4.    <h1>SimpleFormController Demo application</h1>
5.    <hr/><hr/>
6.
7.    <h2>User Details</h2>
8.    User Name    : ${user.name} <br/>
9.    Gender       : ${user.gender} <br/>
10.         Country      : ${user.country} <br/>
11.         About You    : ${user.aboutYou} <br/>
12.         Community    : ${user.community[0]}  ${user.community[1]}
13.                        ${user.community[2]}<br/>
14.         Mailing List: ${user.mailingList}
15.   </center>
16.   </body>
17.   </html>
```

MultiActionController

In Spring MVC application, **MultiActionController** class is used to group related actions into a single controller class, the method handler have to follow the below signature.

```
public (ModelAndView | Map | String | void) actionName(HttpServletRequest,
                        HttpServletResponse [,HttpSession] [,CommandObject]);
```

We can give any name for the method but we have to follow the method signature.



CustomerController.java

```
1. package com.neo.spring.controller;
2.
3. import javax.servlet.http.HttpServletRequest;
4. import javax.servlet.http.HttpServletResponse;
5. import org.springframework.web.servlet.ModelAndView;
6. import
7.     org.springframework.web.servlet.mvc.multiaction.MultiActionController;
8.
9. public class CustomerController extends MultiActionController{
10.
11.         public ModelAndView add(HttpServletRequest request,
12.               HttpServletResponse response) throws Exception {
13.
14.         return new ModelAndView("CustomerPage", "msg","add() method");
15.         }
16.
17.         public ModelAndView delete(HttpServletRequest request,
18.               HttpServletResponse response) throws Exception {
19.
20.         return new ModelAndView("CustomerPage", "msg","delete() method");
21.         }
```

```
22.
23.         public ModelAndView update(HttpServletRequest request,
24.               HttpServletResponse response) throws Exception {
25.
26.         return new ModelAndView("CustomerPage", "msg","update() method");
27.         }
28.
29.         public ModelAndView list(HttpServletRequest request,
30.               HttpServletResponse response) throws Exception {
31.         return new ModelAndView("CustomerPage", "msg","list() method");
32.         }
33.   }
```

spring-beans.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xmlns:p="http://www.springframework.org/schema/p"
5.     xsi:schemaLocation="http://www.springframework.org/schema/beans
6.     http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
7.
8.     <!-- Controllers configuration -->
9.     <bean name="cc" class="com.neo.spring.controller.CustomerController" />
10.
11.
12.         <!-- Handler Mappings configuration-->
13.         <bean id="hm" class="org.springframework.web.servlet.handler
14.                                                .SimpleUrlHandlerMapping">
15.               <property name="mappings">
16.                     <props>
17.                           <prop key="/user/*.sekhar">cc</prop>
18.                     </props>
19.               </property>
20.         </bean>
21.
22.         <!-- View resolver configurations -->
23.         <bean id="viewResolver" class="org.springframework.web.servlet
24.                                        .view.InternalResourceViewResolver">
25.               <property name="prefix" value="/WEB-INF/pages/" />
26.               <property name="suffix" value=".jsp" />
27.         </bean>
28.
29.   </beans>
```

web.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
5.     http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
6.
7.     <servlet>
8.           <servlet-name>front-controller</servlet-name>
9.           <servlet-class>
10.                org.springframework.web.servlet.DispatcherServlet
```

```
11.         </servlet-class>
12.               <init-param>
13.                     <param-name>contextConfigLocation</param-name>
14.                     <param-value>/WEB-INF/spring-beans.xml</param-value>
15.               </init-param>
16.         </servlet>
17.         <servlet-mapping>
18.               <servlet-name>front-controller</servlet-name>
19.               <url-pattern>*.sekhar</url-pattern>
20.         </servlet-mapping>
21.         <welcome-file-list>
22.               <welcome-file>index.jsp</welcome-file>
23.         </welcome-file-list>
24.   </web-app>
```

index.jsp

```
1. <html>
2. <body bgcolor="pink">
3. <center>
4. <h1>Spring MVC MultiActionController Example</h1>
5. <hr><hr>
6.    <a href="user/add.sekhar" >Add customer</a> <br>
7.    <a href="user/delete.sekhar" >Delete customer</a> <br/>
8.    <a href="user/update.sekhar" >Update customer</a> <br/>
9.    <a href="user/list.sekhar" >List customer</a> <br/>
10.   </center>
11.   </body>
12.   </html>
```

CustomerPage.jsp

```
1. <html>
2. <body bgcolor="pink">
3. <center>
4. <h2>Spring MVC MultiActionController Example</h2>
5. <hr><hr>
6. <h2>${msg}</h2>
7. </center>
8. </body>
9. </html>
```

AbstractWizardFormController

SimpleFormController is to handle the single page form submission, which is quite straightforward and easy. But, sometimes, we may need to deal with "**wizard form**", which s**pan the form into multiple pages**, and ask user to fill in the form page by page. The main concern in this wizard form situation is how to store the model data (data fill in by user) and bring it across multiple pages?

Fortunately, Spring MVC comes with **AbstractWizardFormController** class to handle the wizard form easily.
This example shows the use of **AbstractWizardFormController** class to store and bring the form's data across multiple pages, apply validation and display the form's data at the last page.

Wizard Form Pages

Five pages for this demonstration, work as the following sequences :
[User] --> WelcomePage --> Page1 --> Page2 --> Page3 --> ResultPage

With **AbstractWizardFormController**, the page sequence is determined by the "name" of the submit button:

1. _finish: Finish the wizard form.
2. _cancel: Cancel the wizard form.
3. _targetx: Move to the target page, where x is the zero-based page index.
   e.g **_target0**, **_target1** and etc.

Steps to develop:

1. Create a model class to store the form's data (**UserVO**)

2. Write a simple class**(UserController)** and extends the **AbstractWizardFormController**, just override the following methods:
   a. **processFinish**- It fire when user click on the submit button with a name of "**_finish**".
   b. **processCancel** – It fire when user click on the submit button with a name of "**_cancel**".
   c. **formBackingObject** – It use "UserVo" model class to store all the form's data in multiple pages.

3. Write a simple controller **(WelcomeController)** to return a "**WelcomePage**" view.

4. In SimpleFormController, we create a validator class, put all the validation logic inside the **validate()** method, and register the validator to the SimpleFormController declaratively. But, it's a bit different in **AbstractWizardFormController**. First, create a validator class **(UserValidator)**, and also the validation method for each of the page.

5. Write a Properties file**(messages.properties)** to store the error messages.

6. And, in the wizard form controller **(UserController.java)**, override the **validatePage()** by calling the validator manually (no more declaration like simple form controller).

7. In the **validatePage()** method, use a "**switch**" function to determine which page is calling and associated it with the corresponds validator. The page is in 0-indexed.
8. In spring configuration file, Declare the wizard form controller **(UserController.java)**, put all the pages in the correct order and register a validator.

Note
In the "pages" property, the order of the list value is used to define the sequence of the page in the wizard form.

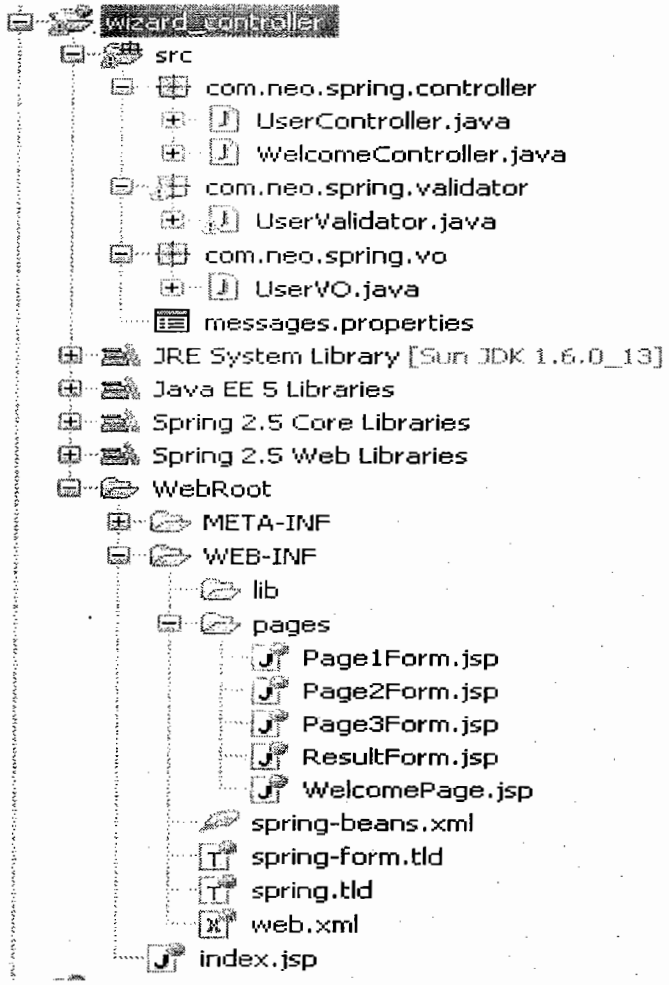
9. **WelcomePage.jsp**
   A welcome page, with a hyperlink to start the wizard form process.

10. **Page1Form.jsp**
    Page 1, with a "username" text box, display error message if any, and contains 2 submit buttons, where:
    a. _target1 – move to page 2.
    b. _cancel – cancel the wizard form process and move it to the cancel page

11. **Page2Form.jsp**
    Page 2, with a "password" field, display error message if any, and contains 3 submit buttons, where:
    a. _target0 – move to page 1.
    b. _target2 – move to page 3.
    c. _cancel – cancel the wizard form process and move it to the cancel page

12. **Page3Form.jsp**
    Page 3, with a "remark" text box, display error message if any, and contains 3 submit buttons, where:
    a. _target1 – move to page 2.
    b. _finish – finish the wizard form process and move it to the finish page.
    c. _cancel – cancel the wizard form process and move it to the cancel page.

13. **ResultForm.jsp**
    Display all the form's data which collected from the previous 3 pages.



UserVO.java

```
1. package com.neo.spring.vo;
2. public class UserVO{
3.
4.     String userName;
5.     String password;
6.     String remark;
7.
8.     public String getRemark() {
9.           return remark;
10.          }
11.          public void setRemark(String remark) {
12.                this.remark = remark;
13.          }
14.          public String getUserName() {
15.                return userName;
16.          }
17.          public void setUserName(String userName) {
18.                this.userName = userName;
19.          }
20.          public String getPassword() {
21.                return password;
22.          }
```

```
23.         public void setPassword(String password) {
24.               this.password = password;
25.         }
26.   }
```

UserController.java

```
1. package com.neo.spring.controller;
2.
3. import javax.servlet.http.HttpServletRequest;
4. import javax.servlet.http.HttpServletResponse;
5. import org.springframework.validation.BindException;
6. import org.springframework.validation.Errors;
7. import org.springframework.web.servlet.ModelAndView;
8. import org.springframework.web.servlet.mvc.AbstractWizardFormController;
9. import com.neo.spring.validator.UserValidator;
10.   import com.neo.spring.vo.UserVO;
11.
12.   public class UserController extends AbstractWizardFormController{
13.
14.         public UserController(){
15.               setCommandClass(UserVO.class);
16.               setCommandName("userForm");
17.         }
18.
19.         @Override
20.         protected ModelAndView processFinish(HttpServletRequest request,
21.                               HttpServletResponse response, Object command,
22.                                     BindException errors) throws Exception {
23.
24.               //Get the data from command object
25.               UserVO user = (UserVO)command;
26.         System.out.println("User is registered with the following data");
27.               System.out.println(user.getUserName());
28.               System.out.println(user.getPassword());
29.               System.out.println(user.getRemark());
30.
31.               //where is the finish page?
32.               return new ModelAndView("ResultForm", "user", user);
33.         }
34.
35.         @Override
36.         protected ModelAndView processCancel(HttpServletRequest request,
37.                               HttpServletResponse response, Object command,
38.                               BindException errors) throws Exception {
39.
40.               //where is the cancel page?
41.               return new ModelAndView("WelcomePage");
42.         }
43.
44.         @Override
45.         protected void validatePage(Object command, Errors errors, int
46.                                                                   page) {
47.
48.               UserValidator validator = (UserValidator) getValidator();
49.
```

```
50.              //page is 0-indexed
51.             switch (page) {
52.              case 0: //if page 1 , go validate with validatePage1Form
53.                          validator.validatePage1Form(command, errors);
54.                          break;
55.              case 1: //if page 2 , go validate with validatePage2Form
56.                          validator.validatePage2Form(command, errors);
57.                          break;
58.              case 2: //if page 3 , go validate with validatePage3Form
59.                          validator.validatePage3Form(command, errors);
60.                          break;
61.              }
62.         }
63.   }
```

WelcomeController.java

```
1. package com.neo.spring.controller;
2.
3. import javax.servlet.http.HttpServletRequest;
4. import javax.servlet.http.HttpServletResponse;
5. import org.springframework.web.servlet.ModelAndView;
6. import org.springframework.web.servlet.mvc.AbstractController;
7.
8. public class WelcomeController extends AbstractController{
9.
10.         @Override
11.         protected ModelAndView handleRequestInternal(HttpServletRequest
12.                 request, HttpServletResponse response) throws Exception {
13.
14.               return new ModelAndView("WelcomePage");
15.         }
16.   }
```

UserValidator.java

```
1. package com.neo.spring.validator;
2.
3. import org.springframework.validation.Errors;
4. import org.springframework.validation.ValidationUtils;
5. import org.springframework.validation.Validator;
6. import com.neo.spring.vo.UserVO;
7.
8. public class UserValidator implements Validator {
9.
10.         public boolean supports(Class clazz) {
11.               // just validate the User instances
12.               return UserVO.class.isAssignableFrom(clazz);
13.         }
14.
15.         // validate page 1, userName
16.         public void validatePage1Form(Object target, Errors errors) {
17.               ValidationUtils.rejectIfEmptyOrWhitespace(errors,
18.              "userName", "required.userName", "Field name is required.");
19.         }
20.
21.         // validate page 2, password
```

```
22.         public void validatePage2Form(Object target, Errors errors) {
23.               ValidationUtils.rejectIfEmptyOrWhitespace(errors,
24.              "password", "required.password", "Field name is required.");
25.         }
26.
27.         // validate page 3, remark
28.         public void validatePage3Form(Object target, Errors errors) {
29.               ValidationUtils.rejectIfEmptyOrWhitespace(errors, "remark",
30.                           "required.remark", "Field name is required.");
31.         }
32.
33.         public void validate(Object target, Errors errors) {
34.
35.               validatePage1Form(target, errors);
36.               validatePage2Form(target, errors);
37.               validatePage3Form(target, errors);
38.         }
39.   }
```

messages.properties

```
1. required.userName = Username is required!
2. required.password = Password is required!
3. required.remark = Remark is required\!
```

spring-beans.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xmlns:p="http://www.springframework.org/schema/p"
5.     xsi:schemaLocation="http://www.springframework.org/schema/beans
6.     http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">
7.
8.     <!-- Controllers configuration -->
9.     <bean id="wc" class="com.neo.spring.controller.WelcomeController" />
10.          <bean id="uc" class="com.neo.spring.controller.UserController">
11.               <property name="pages">
12.                    <list>
13.                         <!-- follow sequence -->
14.                         <value>Page1Form</value> <!-- page1 -->
15.                         <value>Page2Form</value> <!-- page2 -->
16.                         <value>Page3Form</value> <!-- page3 -->
17.                    </list>
18.               </property>
19.               <property name="validator">
20.                   <bean class="com.neo.spring.validator.UserValidator" />
21.               </property>
22.          </bean>
23.
24.          <!-- Handler Mappings configuration-->
25.          <bean id="hm" class="org.springframework.web.servlet.handler
26.                                              .SimpleUrlHandlerMapping">
27.               <property name="mappings">
28.                    <props>
29.                         <prop key="/user.sekhar">uc</prop>
30.                         <prop key="/welcome.sekhar">wc</prop>
```

```
                    </props>
                </property>
        </bean>

        <!-- View resolver configurations -->
        <bean id="viewResolver" class="org.springframework.web.servlet
                                        .view.InternalResourceViewResolver">
                <property name="prefix" value="/WEB-INF/pages/" />
                <property name="suffix" value=".jsp" />
        </bean>

        <!-- Register messages.properties for validation error message-->
        <bean id="messageSource" class="org.springframework.context
                                    .support.ResourceBundleMessageSource">
                <property name="basename" value="messages" />
        </bean>

    </beans>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

    <servlet>
        <servlet-name>front-controller</servlet-name>
        <servlet-class>
                org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
            <init-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>/WEB-INF/spring-beans.xml</param-value>
            </init-param>
        </servlet>
        <servlet-mapping>
            <servlet-name>front-controller</servlet-name>
            <url-pattern>*.sekhar</url-pattern>
        </servlet-mapping>
        <welcome-file-list>
            <welcome-file>index.jsp</welcome-file>
        </welcome-file-list>
    </web-app>
```

index.jsp

```
<html>
<body bgcolor="pink">
    <center>
        <h2>Handling multipage forms in Spring MVC</h2>
        <hr/> <hr/>
        <a href="welcome.sekhar">Welcome page</a>
    </center>
</body>
</html>
```

welcomePage.jsp

```
<html>
<body bgcolor="pink">
    <center>
          <h2>Handling multipage forms in Spring MVC</h2>
          <hr/> <hr/>
          Click here to start playing -
          <a href="user.sekhar">AbstractWizardFormController example</a>
    </center>
</body>
    </html>
```

page1Form.jsp

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<html>
<head>
<style>
.error {
    color: #ff0000;
}
.errorblock{
    color: #000;
          background-color: #ffEEEE;
          border: 3px solid #ff0000;
          padding:8px;
          margin:16px;
    }
    </style>
    </head>

    <body bgcolor="pink" >
    <center>
    <h2>Page1Form.jsp</h2>
    <hr/><hr/>
    <form:form method="POST" commandName="userForm">
    <form:errors path="*" cssClass="errorblock" element="div"/>
    <table>
          <tr>
                <td>Username : </td>
                <td><form:input path="userName" /></td>
                <td><form:errors path="userName" cssClass="error" /></td>
          </tr>

          <tr>
                <td colspan="3">
                      <input type="submit" value="Next" name="_target1" />
                      <input type="submit" value="Cancel" name="_cancel" />
                </td>
          </tr>

    </table>
    </form:form>
    </center>
    </body>
```

```
42. </html>
```

page2Form.jsp

```
1. <%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
2. <html>
3. <head>
4. <style>
5. .error {
6.    color: #ff0000;
7. }
8. .errorblock{
9.    color: #000;
10.         background-color: #ffEEEE;
11.         border: 3px solid #ff0000;
12.         padding:8px;
13.         margin:16px;
14.   }
15.   </style>
16.   </head>
17.
18.   <body bgcolor="pink" >
19.   <center>
20.   <h2>Page2Form.jsp</h2>
21.   <hr/><hr>
22.   <form:form method="POST" commandName="userForm">
23.   <form:errors path="*" cssClass="errorblock" element="div"/>
24.
25.   <table>
26.         <tr>
27.               <td>Password : </td>
28.               <td><form:password path="password" /></td>
29.               <td><form:errors path="password" cssClass="error" /></td>
30.         </tr>
31.         <tr>
32.               <td colspan="3">
33.               <input type="submit" value="Previous" name="_target0" />
34.               <input type="submit" value="Next" name="_target2" />
35.               <input type="submit" value="Cancel" name="_cancel" />
36.               </td>
37.         </tr>
38.   </table>
39.   </form:form>
40.   </center>
41.   </body>
42.   </html>
```

page3Form.jsp

```
1. <%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
2. <html>
3. <head>
4. <style>
5. .error {
6.    color: #ff0000;
7. }
8. .errorblock{
```

```
9.    color: #000;
10.         background-color: #ffEEEE;
11.         border: 3px solid #ff0000;
12.         padding:8px;
13.         margin:16px;
14.   }
15.   </style>
16.   </head>
17.
18.   <body bgcolor="pink" >
19.   <center>
20.   <h2>Page3Form.jsp</h2>
21.   <hr/><hr>
22.   <form:form method="POST" commandName="userForm">
23.   <form:errors path="*" cssClass="errorblock" element="div"/>
24.   <table>
25.         <tr>
26.               <td>Remark : </td>
27.               <td><form:input path="remark" /></td>
28.               <td><form:errors path="remark" cssClass="error" /></td>
29.         </tr>
30.         <tr>
31.               <td colspan="3">
32.               <input type="submit" value="Previous" name="_target1" />
33.               <input type="submit" value="Finish" name="_finish" />
34.               <input type="submit" value="Cancel" name="_cancel" />
35.         </td>
36.         </tr>
37.   </table>
38.   </form:form>
39.   </center>
40.   </body>
41.   </html>
```

resultForm.jsp

```
1. <html>
2. <body bgcolor="pink">
3. <center>
4. <h2>ResultForm.jsp</h2>
5. <hr/><hr/>
6. <table>
7.    <tr>
8.          <td>UserName :</td><td>${user.userName}</td>
9.    </tr>
10.         <tr>
11.               <td>Password :</td><td>${user.password}</td>
12.         </tr>
13.         <tr>
14.               <td>Remark :</td><td>${user.remark}</td>
15.         </tr>
16.   </table>
17.   </center>
18.   </body>
19.   </html>
```

Struts Spring Integration

Approach 1:

Step 1: Create a project and add the capabilities(jars) of struts and spring(web libraries and core libraries).

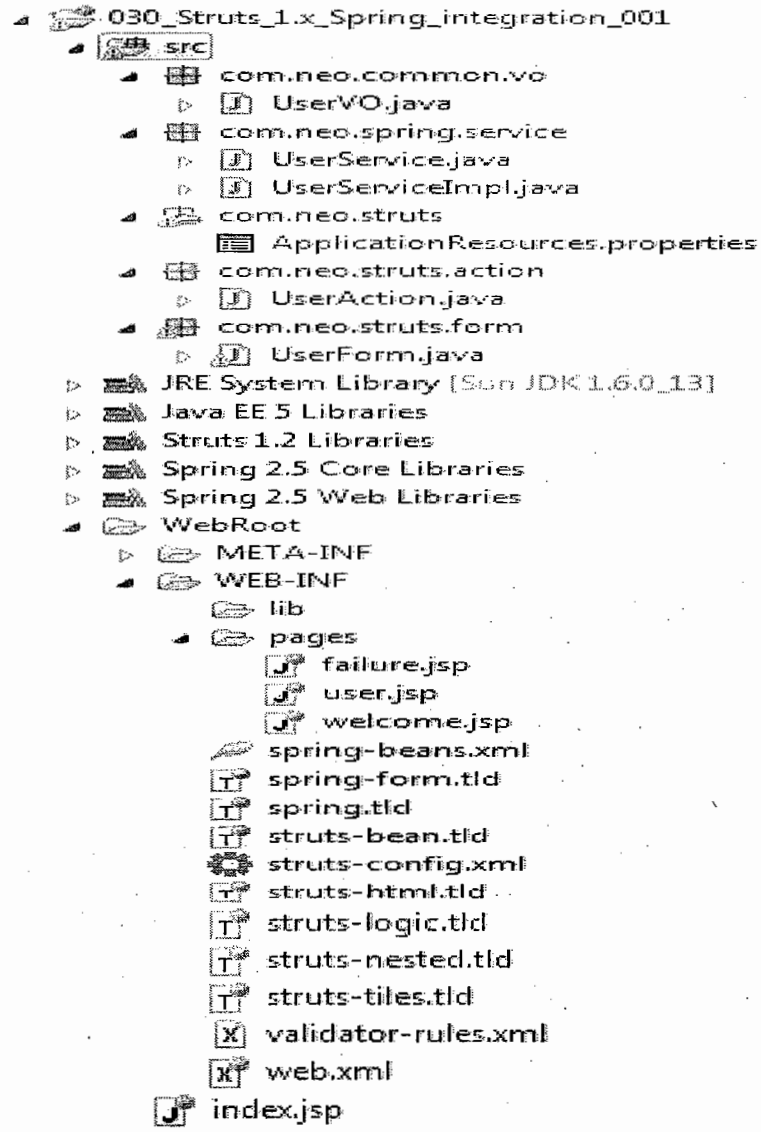
Step 2: Develop spring beans, and configure them in spring configuration file.

Step 3: Configure spring given plug-in in struts-config.xml file. Pass spring configuration file name and location, to the spring given plug-in using <set-property> sub tag of <plug-in> tag.

```
<plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">
        <set-property property="contextConfigLocation"
                value="/WEB-INF/spring-beans.xml" />
</plug-in>
```

Step 4: Develop Action class, which extends ActionSupport given by spring framework, to get spring container in action class and to communicate with any spring beans. That support is given in ActionSupport. (ActionSupport is the child class of Action)

Step 5: Get the spring container by calling **getWebApplicationContext()**. So using container get the beans configured in spring configuration file. And call the business methods on spring beans.



UserVO.java

```
1. package com.neo.common.vo;
2.
3. public class UserVO {
4.     private String name;
5.
6.     public String getName() {
7.           return name;
8.     }
9.
10.          public void setName(String name) {
11.                this.name = name;
12.          }
13.
14.    }
```

UserService.java

```
1. package com.neo.spring.service;
```

```
2.
3. import com.neo.common.vo.UserVO;
4.
5. public interface UserService {
6.     String sayWelcome(UserVO userVO);
7. }
8.
9. UserServiceImpl.java
10.    package com.neo.spring.service;
11.
12.    import com.neo.common.vo.UserVO;
13.
14.    public class UserServiceImpl implements UserService {
15.          public String sayWelcome(UserVO userVO) {
16.                return "Welcome : " + userVO.getName();
17.          }
18.    }
```

UserAction.java

```
1. package com.neo.struts.action;
2.
3. import javax.servlet.http.HttpServletRequest;
4. import javax.servlet.http.HttpServletResponse;
5.
6. import org.apache.commons.beanutils.BeanUtils;
7. import org.apache.struts.action.ActionForm;
8. import org.apache.struts.action.ActionForward;
9. import org.apache.struts.action.ActionMapping;
10.    import org.springframework.web.context.WebApplicationContext;
11.    import org.springframework.web.struts.ActionSupport;
12.
13.    import com.neo.common.vo.UserVO;
14.    import com.neo.spring.service.UserService;
15.    import com.neo.struts.form.UserForm;
16.
17.    public class UserAction extends ActionSupport {
18.          @Override
19.          public ActionForward execute(ActionMapping mapping, ActionForm form,
20.                      HttpServletRequest request, HttpServletResponse response)
21.                      throws Exception {
22.
23.                UserForm userForm = (UserForm) form;
24.                UserVO userVo = new UserVO();
25.                BeanUtils.copyProperties(userVo, userForm);
26.                WebApplicationContext context = getWebApplicationContext();
27.                UserService service = (UserService) context.getBean("userService");
28.                String welcomeMessage = service.sayWelcome(userVo);
29.                request.setAttribute("welcomeMsg", welcomeMessage);
30.                return mapping.findForward("success");
31.          }
32.    }
```

UserForm.java

```
1. package com.neo.struts.form;
2.
```

```
3. import org.apache.struts.action.ActionForm;
4.
5. public class UserForm extends ActionForm {
6.    private String name;
7.
8.    public String getName() {
9.          return name;
10.         }
11.
12.         public void setName(String name) {
13.               this.name = name;
14.         }
15.
16.   }
```

failure.jsp

```
1. <HTML>
2. <BODY BGCOLOR="red">
3.      <H1> Struts-spring integration application</H1>
4.     <H1> FAILED</H1>
5. </BODY>
6. </HTML>
```

user.jsp

```
1. <%@ taglib  prefix="html"  uri="/WEB-INF/struts-html.tld" %>
2. <html:html>
3.      <BODY BGCOLOR="pink">
4.         <CENTER>
5.           <H1> Struts-spring integration application</H1>
6.                <hr/><hr/>
7.                <html:form  action="welcome" method="post" >
8.             NAME : <html:text  property="name" /><BR><BR>
9.            <html:submit>Get Welcome</html:submit>
10.               </html:form>
11.          </CENTER>
12.          </BODY>
13.   </html:html>
```

welcome.jsp

```
1. <HTML>
2. <BODY BGCOLOR="pink">
3.    <center>
4.      <H1> Struts-spring integration application</H1>
5.      <hr/> <hr/>
6.          <h2> Welcome Message : ${welcomeMsg}</h2>
7.     </center>
8. </BODY>
9. </HTML>
```

spring-beans.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:p="http://www.springframework.org/schema/p"
```

```
4.    xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
5.
6.    <bean id="userService" class="com.neo.spring.service.UserServiceImpl"></bean>
7.
8. </beans>
```

struts-config.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts
3. Configuration 1.2//EN" "http://struts.apache.org/dtds/struts-config_1_2.dtd">
4.
5. <struts-config>
6.    <form-beans>
7.          <form-bean name="userForm" type="com.neo.struts.form.UserForm" />
8.    </form-beans>
9.    <action-mappings>
10.             <action path="/user" forward="/WEB-INF/pages/user.jsp"></action>
11.             <action path="/welcome" type="com.neo.struts.action.UserAction"
12.                   name="userForm">
13.                   <forward name="success" path="/WEB-INF/pages/welcome.jsp" />
14.                   <forward name="failure" path="/WEB-INF/pages/failure.jsp" />
15.             </action>
16.       </action-mappings>
17.       <message-resources parameter="com.neo.struts.ApplicationResources" />
18.       <plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">
19.             <set-property property="contextConfigLocation"
20.                         value="/WEB-INF/spring-beans.xml" />
21.       </plug-in>
22.
23.   </struts-config>
```

index.jsp

```
<jsp:forward page="user.do"/>
```

Approach 2:

Step 1: Create a project and add the capabilities(jars) of struts and spring(web libraries and core libraries).

Step 2: Develop spring beans, and configure them in spring configuration file.

Step 3: Configure spring given plug-in in struts-config.xml file. Pass spring configuration file name and location, to the spring given plug-in using <set-property> sub tag of <plug-in> tag.

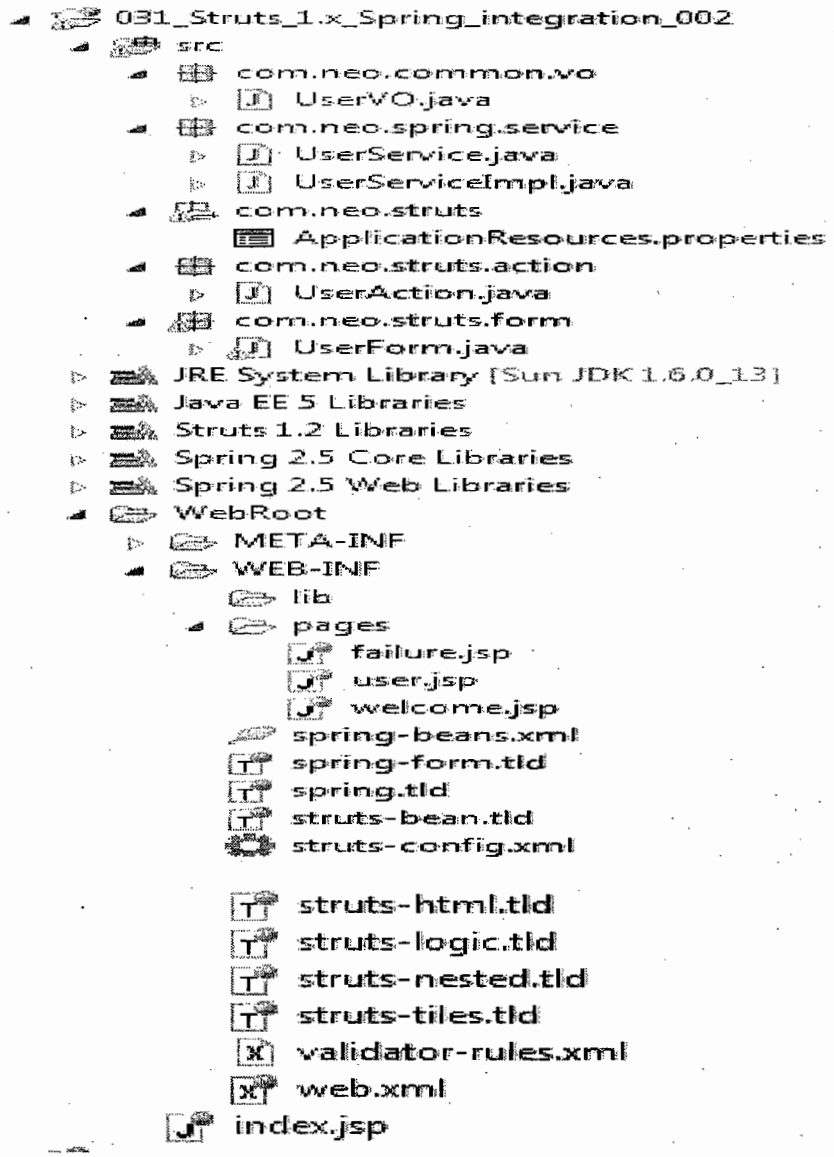
<plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">

<set-property property="contextConfigLocation"

```
                value="/WEB-INF/spring-beans.xml" />
</plug-in>
```

Step 4: Develop Action class, configure in struts-config.xml and spring configuration file. And inject service class to action class.

Step 5: Configure spring given org.springframework.web.struts.DelegatingRequestProcessor(child of RequestProcessor) class controller in struts-config.xml.



UserVO.java

```
1. package com.neo.common.vo;
2. public class UserVO {
3.    private String name;
```

```
4.
5.    public String getName() {
6.          return name;
7.    }
8.
9.    public void setName(String name) {
10.         this.name = name;
11.   }
12.   }
```

UserService.java

```
1. package com.neo.spring.service;
2. import com.neo.common.vo.UserVO;
3.
4. public interface UserService {
5.    String sayWelcome(UserVO userVO);
6. }
```

UserServiceImpl.java

```
1. package com.neo.spring.service;
2. import com.neo.common.vo.UserVO;
3.
4. public class UserServiceImpl implements UserService {
5.    public String sayWelcome(UserVO userVO) {
6.          return "Welcome : " + userVO.getName();
7.    }
8. }
```

UserAction.java

```
1. package com.neo.struts.action;
2.
3. import javax.servlet.http.HttpServletRequest;
4. import javax.servlet.http.HttpServletResponse;
5. import org.apache.commons.beanutils.BeanUtils;
6. import org.apache.struts.action.Action;
7. import org.apache.struts.action.ActionForm;
8. import org.apache.struts.action.ActionForward;
9. import org.apache.struts.action.ActionMapping;
10.   import com.neo.common.vo.UserVO;
11.   import com.neo.spring.service.UserService;
12.   import com.neo.struts.form.UserForm;
13.
14.   public class UserAction extends Action {
15.         private UserService service;
16.
17.         public void setService(UserService service) {
18.               this.service = service;
19.         }
20.
21.         @Override
22.         public ActionForward execute(ActionMapping mapping, ActionForm form,
23.                     HttpServletRequest request, HttpServletResponse response)
24.                     throws Exception {
```

```
25.
26.                 UserForm userForm = (UserForm) form;
27.                 UserVO userVo = new UserVO();
28.                 BeanUtils.copyProperties(userVo, userForm);
29.                 String welcomeMessage = service.sayWelcome(userVo);
30.                 request.setAttribute("welcomeMsg", welcomeMessage);
31.                 return mapping.findForward("success");
32.           }
33.    }
```

UserForm.java

```
1. package com.neo.struts.form;
2.
3. import org.apache.struts.action.ActionForm;
4.
5. public class UserForm extends ActionForm {
6.     private String name;
7.
8.     public String getName() {
9.           return name;
10.          }
11.
12.          public void setName(String name) {
13.                this.name = name;
14.          }
15.
16.    }
```

failure.jsp

```
1. <HTML>
2. <BODY BGCOLOR="red">
3.        <H1> Struts-spring integration application</H1>
4.       <H1> FAILED</H1>
5. </BODY>
6. </HTML>
```

user.jsp

```
1. <%@ taglib  prefix="html"  uri="/WEB-INF/struts-html.tld" %>
2. <html:html>
3.       <BODY BGCOLOR="pink">
4.          <CENTER>
5.            <H1> Struts-spring integration application</H1>
6.                 <hr/><hr/>
7.                 <html:form  action="welcome" method="post" >
8.              NAME : <html:text  property="name" /><BR><BR>
9.             <html:submit>Get Welcome</html:submit>
10.                </html:form>
11.           </CENTER>
12.           </BODY>
13.    </html:html>
```

welcome.jsp

```
1. <HTML>
2. <BODY BGCOLOR="pink">
3.     <center>
```

```
4.      <H1> Struts-spring integration application</H1>
5.      <hr/> <hr/>
6.          <h2> Welcome Message : ${welcomeMsg}</h2>
7.     </center>
8. </BODY>
9. </HTML>
```

spring-beans.xml

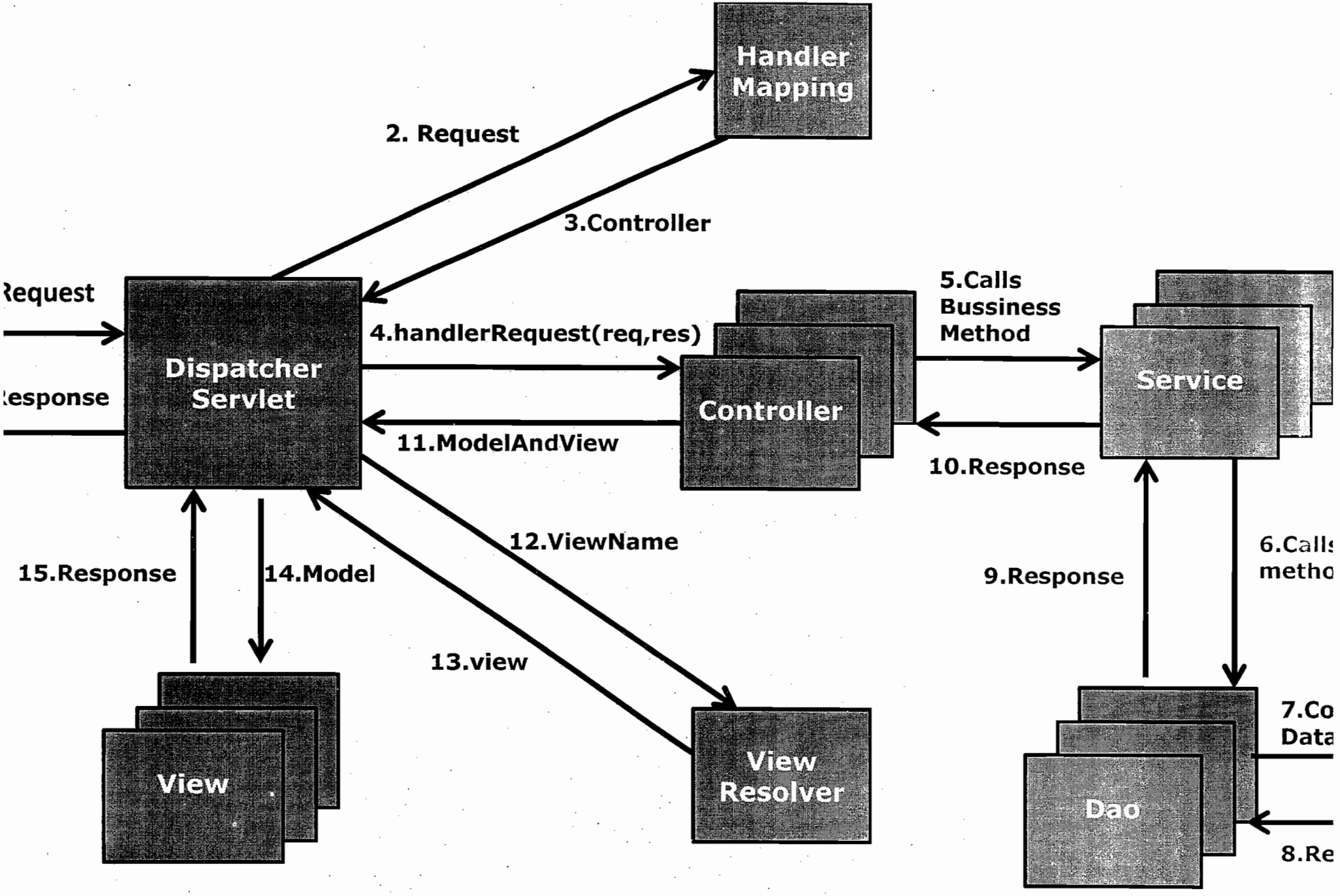
```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <beans xmlns="http://www.springframework.org/schema/beans"
3.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4. xmlns:p="http://www.springframework.org/schema/p"
5.    xsi:schemaLocation="http://www.springframework.org/schema/beans
6. http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7.
8.    <bean id="serviceImpl" class="com.neo.spring.service.UserServiceImpl"></bean>
9.
10.         <bean name="/welcome" class="com.neo.struts.action.UserAction">
11.              <property name="service" ref="serviceImpl"></property>
12.   </bean>
13.
14.   </beans>
```

struts-config.xml

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts
3. Configuration 1.2//EN" "http://struts.apache.org/dtds/struts-config_1_2.dtd">
4. <struts-config>
5.    <form-beans>
6.         <form-bean name="userForm" type="com.neo.struts.form.UserForm" />
7.    </form-beans>
8.    <action-mappings>
9.         <action path="/user" forward="/WEB-INF/pages/user.jsp"></action>
10.             <action path="/welcome" type="com.neo.struts.action.UserAction"
11.                  name="userForm">
12.                  <forward name="success" path="/WEB-INF/pages/welcome.jsp" />
13.                  <forward name="failure" path="/WEB-INF/pages/failure.jsp" />
14.             </action>
15.        </action-mappings>
16.
17.        <controller
18.        processorClass="org.springframework.web.struts.DelegatingRequestProcessor" />
19.
20.        <message-resources parameter="com.neo.struts.ApplicationResources" />
21.
22.        <plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">
23.             <set-property property="contextConfigLocation"
24.                           value="/WEB-INF/spring-beans.xml" />
25.        </plug-in>
26.
27.   </struts-config>
```

index.jsp

```
<jsp:forward page="user.do"/>
```



Service
Handler Mapping
Controller
View Resolver
Dispatcher Servlet
View