Handler Mapping

Define how web request (URL) maps to the Controller handlers.

BeanNameUrlHandlerMapping

Maps the requested URL to the name of the cocntroller.

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

<bean name="/welcome.htm" class="com.h2k.controller.WelcomeController" />

/welcome.htm is requested, DispatcherServlet will forward the request to the “WelcomeController“.

ControllerClassNameHandlerMapping

Uses convention to map the requested URL to Controller.

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

<bean class="com.h2k.controller.WelcomeController" />

Now, Spring MVC is mapping the requested URL by following convention:

```
WelcomeController -> /welcome*
```

SimpleUrlHandlerMapping

Allow developer to specify the mapping of URL patterns and handler mappings explicitly.

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

```
  </property>
</bean>

<bean id="welcomeController"  class="com.h2k.controller.WelcomeController" />
```

Both are defined the same handler mappings.

1. /welcome.htm –> welcomeController.
2. /{anything}/welcome.htm –> welcomeController.

Configure the handler mapping priority

If multiple view handler mappings are applied, you have to declare priority to avoid conflict issue. You can do that with the help of Order property:

```
<property name="order" value="0" />
```

Controller

Controller class to handle the web request.

MultiActionController:

1. Controller class should extend MultiActionController
2. Multiple actions can be configured in same controller.

```
public class CustomerController extends MultiActionController{
 public ModelAndView add(HttpServletRequest request,
          HttpServletResponse response) throws Exception {
          return new ModelAndView("CustomerPage", "msg","add() method");
 }

 public ModelAndView delete(HttpServletRequest request,
          HttpServletResponse response) throws Exception {
          return new ModelAndView("CustomerPage", "msg","delete() method");
 }
}
```

With **ControllerClassNameHandlerMapping** configured.
Now, the requested URL will map to the method name in the following patterns :

1. **Customer**Controller –> **/customer/***
2. /customer/**add**.htm –> **add()**
3. /customer/**delete**.htm –> **delete()**

With InternalPathMethodNameResolver

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

  <bean class="com.h2k.controller.CustomerController">
    <property name="methodNameResolver">
<bean
class="org.springframework.web.servlet.mvc.multiaction.InternalPathMethodNameResolver">
            <property name="prefix" value="test" />
            <property name="suffix" value="Customer" />
          </bean>
    </property>
  </bean>
</beans>
```

MultiActionController Annotation

Now, the URL will map to the method name in the following pattern:

1. **Customer**Controller –> **/customer/***
2. /customer/**add**.htm –> test**add**Customer()
3. /customer/**delete**.htm –> test**delete**Customer()

```
@Controller
public class CustomerController{

@RequestMapping("/customer/add.htm")
public ModelAndView add(HttpServletRequest request,
        HttpServletResponse response) throws Exception {
        return new ModelAndView("CustomerAddView");
}

@RequestMapping("/customer/delete.htm")
public ModelAndView delete(HttpServletRequest request,
        HttpServletResponse response) throws Exception {
        return new ModelAndView("CustomerDeleteView");
}

}
```

Now, the URL will map to the method name in the following patterns :

1. /customer/add.htm –> add() method
2. /customer/delete.htm –> delete() method

PropertiesMethodNameResolver

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

<bean class="com.h2k.controller.CustomerController">
<property name="methodNameResolver">
   <bean class="org.springframework.web.servlet.mvc.multiaction.PropertiesMethodNameResolver">
  <property name="mappings">
  <props>
     <prop key="/customer/a.htm">add</prop>
     <prop key="/customer/b.htm">update</prop>
     <prop key="/customer/whatever.htm">add</prop>
  </props>
  </property>
  </bean>
</property>
</bean>
```

Now, the URL will map to the method name in the following pattern :

1. /customer/a.htm –> add() method
2. /customer/b.htm –> update() method
3. /customer/whatever.htm –> add() method

ParameterMethodNameResolver

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

<bean class="com.h2k.controller.CustomerController">
<property name="methodNameResolver">
<bean class="org.springframework.web.servlet.mvc.multiaction.ParameterMethodNameResolver">
  <property name="paramName" value="action"/>
</bean>
</property>
</bean>
```

Now, the URL will map to the method name via the “action” request parameter name :

1. /customer/*.htm**?action=add** –> add() method
2. /customer/whatever.htm**?action=add** –> add() method
3. /customer/*.htm**?action=delete** –> delete() method

ParameterizableViewController

No controller class is required, just declared the **ParameterizableViewController** bean and specify the view name through the “**viewName**” property. Additionally, you have to define an explicit mapping for it.

```
<bean name="welcomeController"
   class="org.springframework.web.servlet.mvc.ParameterizableViewController">
   <property name="viewName" value="WelcomePage" />
</bean>
```

ViewResolver

InternalResourceViewResolver

```
<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>
```

Now, Spring will resolve the view’s name “**WelcomePage**” in the following way :

prefix + view name + suffix = /WEB-INF/pages/**WelcomPage**.jsp

XmlViewResolver

In Spring MVC, XmlViewResolver is used to resolve “view name” 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:

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

The "**view bean**" is just a normal Spring bean declared in the Spring's bean configuration file, 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.

```
<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>
```

ResourceBundleViewResolver

In Spring MVC, ResourceBundleViewResolver is used to resolve "view named" based on view beans in ".properties" file.

By default, ResourceBundleViewResolver will loads the view beans from file views.properties, which located at the root of the project class path. However, this location can be overridden through the "basename" property, for example,

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

Declare each view bean as a normal resource bundle style (key & message), where

"WelcomePage" is the view name to match.
".(class)" is the type of view.
".url" is the view's URL location.

File : spring-views.properties

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

Note:
Put this “spring-views.properties” file on your project class path.