Spring Web Flow and MVC Integration

Summary

This chapter shows how to integrate Web Flow into a Spring MVC web application. The booking-mvc sample application is a good reference for Spring MVC with Web Flow. This application is a simplified travel site that allows users to search for and book hotel rooms.

Description

The first step to using Spring MVC is to configure the DispatcherServlet in web.xml.

Configuring web.xml

The first step to using Spring MVC is to configure the DispatcherServlet in web.xml. You typically do this once per web application.

DispatcherServlet registers one for each web application.

In this example, it is set to receive all requests that starts with /spring/. contextConfigLocation can be set using init-param.
web.xml

```
<servlet>
	<servlet-name>Spring MVC Dispatcher Servlet</servlet-name>
	<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
	<init-param>
		<param-name>contextConfigLocation</param-name>
		<param-value>/WEB-INF/web-application-config.xml</param-value>
	</init-param>
</servlet>
<servlet-mapping>
	<servlet-name>Spring MVC Dispatcher Servlet</servlet-name>
	<url-pattern>/spring/*</url-pattern>
</servlet-mapping>
```

Dispatching to flows

The DispatcherServlet maps requests for application resources to handlers. A flow is one type of handler.

Registering the FlowHandlerAdapter

First of all, control flow in Spring MVC by defining FlowHandlerAdapter Bean and set flowExecutor bean with property(flowExecutor).

```
<!-- Enables FlowHandler URL mapping -->
<bean class="org.springframework.webflow.mvc.servlet.FlowHandlerAdapter">
	<property name="flowExecutor" ref="flowExecutor" />
</bean>
```

Defining flow mapping

Once flow handling is enabled, the next step is to map specific application resources to your flows. The simplest way to do this is to define a FlowHandlerMapping:

```
<!-- Maps request paths to flows in the flowRegistry; e.g. a path of /hotels/booking looks for a flow
with id "hotels/booking" -->
```

```
<bean class="org.springframework.webflow.mvc.servlet.FlowHandlerMapping">
	<property name="flowRegistry" ref="flowRegistry" />
</bean>
```

This setting enables the Dispatcher to map the application resource path to the flow registered in flow registry.
For example, /hotels/booking request gives request to flow with Flow ID of hotels/booking.
If it fails to find relevant ID in flow registry, find the next handler mapping according to order of Dispatcher and if there is next one, have "noHandlerFound"(?) response return.

Flow handing workflow

When a valid flow mapping is found, the FlowHandlerAdapter figures out whether to start a new execution of that flow or resume an existing execution based on information present the HTTP request. There are a number of defaults related to starting and resuming flow executions the adapter employs:

- HTTP request parameters are made available in the input map of all starting flow executions.
- When a flow execution ends without sending a final response, the default handler will attempt to start a new execution in the same request.
- Unhandled exceptions are propagated to the Dispatcher unless the exception is a NoSuchFlowExecutionException. The default handler will attempt to recover from a NoSuchFlowExecutionException by starting over a new execution.

For more details, refer to FlowHandlerAdapter in the SWF JavaDoc.

Implementing custom FlowHandler

FlowHandler is the extension point that can be used to customize how flows are executed in a HTTP servlet environment. A FlowHandler is used by the FlowHandlerAdapter and is responsible for:

- Returning the id of a flow definition to execute
- Creating the input to pass new executions of that flow as they are started
- Handling outcomes returned by executions of that flow as they end
- Handling any exceptions thrown by executions of that flow as they occur

These responsibilities are illustrated in the definition of the org.springframework.mvc.servlet.FlowHandler interface:

```
 public interface FlowHandler {

	public String getFlowId();

	public MutableAttributeMap createExecutionInputMap(HttpServletRequest request);

	public String handleExecutionOutcome(FlowExecutionOutcome outcome, HttpServletRequest
request, HttpServletResponse response);

	public String handleException(FlowException e,	HttpServletRequest request,
HttpServletResponse response);
}
```

If implementing FlowHandler, inherit AbstractFlowHandler. All operations are selective and can be implemented only when necessary.
If not implemented, contents of default implementation (implemented in AbstractFlowHandler) is applied.
In particular, implementation can be examined in the following cases.

- getFlowId(HttpServletRequest) re-definition: When Flow id cannot be obtained directly at other places of HTTP. In general, get the path information from the URI of request. For example, http://localhost/app/hotels/booking?hotelId=1 is mapped to flow id of hotels/booking

- createExecutionInputMap(HttpServletRequest) redefinition: if directly extracting flow input parameter in HttpServletRequest. By default, all request parameters are transferred to flow input parameter.
- handleExecutionOutcome redefinition: if required to control results of flow execution directly. Default action is redirected to URL of last flow to restart new execution of Flow
- handleExeception redefinition: if required to adjust the execution of flow uncontrolled in details. By default, uncontrolled Exception is sent to Spring MVC ExceptionResolver again.

Example FlowHandler

A common interaction pattern between Spring MVC And Web Flow is for a Flow to redirect to a @Controller when it ends. FlowHandlers allow this to be done without coupling the flow definition itself with a specific controller URL. An example FlowHandler that redirects to a Spring MVC Controller is shown below:

```
public class BookingFlowHandler extends AbstractFlowHandler {
          public String handleExecutionOutcome(FlowExecutionOutcome outcome, HttpServletRequest
request,
                                                    HttpServletResponse response) {
                    if (outcome.getId().equals("bookingConfirmed")) {
                              return "/booking/show?bookingId=" +
outcome.getOutput().get("bookingId");
                    } else {
                              return "/hotels/index";
                    }
          }
}
```

If the flow id as a result of flow is bookingConfirmed in redefined handleExecutionOutcome method, send to specific URL(/booking/show?bookingId=…).
If flow id is different from bookingConfirmed, go to URL for /hotels/index.

Deploying a custom FlowHandler

To install a custom FlowHandler, simply deploy it as a bean. The bean name must match the id of the flow the handler should apply to.

```
<bean name="hotels/booking"
class="org.springframework.webflow.samples.booking.BookingFlowHandler"/>
```

Through this setting, approach to /hotels/booking executes hotels/booking using the custom handler of BookingFlowHandler.
At the time when booking flow finishes, handleExecutionOutcome of BookingFlowHandler is executed and resent to the appropriate controller according to String(URL) results.

FlowHandler Redirects

A FlowHandler handling a FlowExecutionOutcome or FlowException returns a String to indicate the resource to redirect to after handling. In the previous example, the BookingFlowHandler redirects to the booking/show resource URI for bookingConfirmed outcomes, and the hotels/index resource URI for all other outcomes.

By default, returned resource locations are relative to the current servlet mapping. This allows for a flow handler to redirect to other Controllers in the application using relative paths. In addition, explicit redirect prefixes are supported for cases where more control is needed.

The explicit redirect prefixes supported are:

- servletRelative: - redirect to a resource relative to the current servlet

- contextRelative: - redirect to a resource relative to the current web application context path
- serverRelative: - redirect to a resource relative to the server root
- http:// or https:// - redirect to a fully-qualified resource URI

These same redirect prefixes are also supported within a flow definition when using the externalRedirect: directive in conjunction with a view-state or end-state; for example, view="externalRedirect:http://springframework.org"

View Resolution

Web Flow 2 maps the file in the directory where there is a flow file and the view identifier selected if not separately designated. Existing spring MVC+Web Flow application processes the mapping processing of external ViewResolver already. Therefore, to continue to use existing resolver and to avoid the changing of packaging method of existing view, perform the next setting.

```
<webflow:flow-registry id="flowRegistry"     flow-builder-services="flowBuilderServices">
          <webflow:location path="/WEB-INF/hotels/booking/booking.xml" />
</webflow:flow-registry>

<webflow:flow-builder-services id="flowBuilderServices"         view-
factorycreator="mvcViewFactoryCreator" />

<bean id="mvcViewFactoryCreator"
class="org.springframework.webflow.mvc.builder.MvcViewFactoryCreator">
          <property name="viewResolvers" ref="myExistingViewResolverToUseForFlows" />
</bean>
```

MvcViewFactoryCreator enables to use Spring MVC view system(here, “myExistingViewResolverToUseForFlows”) in Spring Web Flow.
Booking Hotels sample is set as follows.(it is set to use tilesViewResolver.)

```
<bean id="mvcViewFactoryCreator"
class="org.springframework.webflow.mvc.builder.MvcViewFactoryCreator">
          <property name="viewResolvers" ref="tilesViewResolver"/>
</bean>
```

In addition, if value of useSpringBinding is true, can perform data binding using the unique BeanWrapper of Spring MVC.

Signaling an event from a View

If flow stops temporarily in view-state, resend the user to relevant execution URL and wait for user event to start again. In this section, let’s examine the method of generating the event in HTML based view created by template engine like JSP, Velocity or Freemarker.

Using a named HTML button to signal an event

Following shows 2 buttons in one form that generates proceed and cancle event.

```
<input type="submit" name="_eventId_proceed" value="Proceed" />
<input type="submit" name="_eventId_cancel" value="Cancel" />
```

When the button is selected, SWF looks for the request parameter that starts with _eventId, cut out the area and use the remaining character string as id.
_eventId_proceed is proceeded. So that various events can be created in same form.

Using a hidden HTML form parameter to signal an event

Let’s do the following to generate proceed event when form is submitted.

```
<input type="submit" value="Proceed" />
<input type="hidden" name="_eventId" value="proceed" />
```

Here, find the value to _eventId parameter and use the relevant value with event id.
This method should be considered when there is only 1 event that can transfer to the form.

Using a HTML link to signal an event

```
<a href="${flowExecutionUrl}&_eventId=cancel">Cancel</a>
```

The parameter identification order is "eventId" ⇒ "_eventId" ⇒ no.

Reference

- Spring Web Flow reference 2.0.x
- Spring Web-Flow Framework Reference beta with Korean (by Park Chan Wook)