CHAPTER 11

Building Applications with Spring Web Flow

In the previous chapter on Spring Web Flow, we introduced you to Web Flow’s main features and what it can do for you. We also covered what a basic Web Flow configuration looks like and how it can be integrated with Spring MVC. We saw some basic elements that are part of almost every flow, such as the view state and the evaluate action. And we ended by converting our Bookstore to a basic Web Flow-enabled application.

In this chapter, we will continue to look at more features of Web Flow. For example, we will see all the features required to turn the bookstore in a real-world application. We will be starting from the bookstore as we left it in the previous chapter, which we will be refactoring as we go along. The general approach of this chapter is a mix: we will introduce you to some new features first, explaining them in detail. Next, we will see how these concepts can be applied by refactoring the existing application or using these features to modify the bookstore.

We have chosen this approach, so that the table of contents can continue to be referenced. When working with Web Flow, you probably want to browse back to certain functionality and read about all its details. If we were to mix everything together, this would become difficult. We will tackle the sample in two stages, to make everything more digestible.

Important Web Flow Concepts

Now that you have seen some basic Web Flow coverage, we want to explain some specific new Web Flow concepts and terminology, as well as to expand on some concepts introduced in Chapter 10. Specifically, we will expand the definition of “flow” introduced in the Chapter 10. We will also take a closer look at scopes and how they work, and then introduce and define the implicit objects available for Web Flow expressions.

Flow Definition

In a flow definition, we can have five states, two of which we have already seen: the view state and the end state. The remaining states are the action state, decision state, and the subflow state. We will go over these in more detail in this chapter. However, in this section, we want to give a short introduction to subflows, since we will refer to them before covering them fully in the “Subflows” section. The concept of a subflow is pretty easy. For example, the structure and setup of a subflow is not too different from any other flow; it is defined in its own flow definition XML, and it has its own flow id.

Sometimes a given flow is started from another flow, rather than starting it at the top level by referring to it directly. We call such a flow a subflow. This process creates a parent/child relationship between the two flows. As we will see later, this relationship implies that the two flows can share state
using the conversation scope, as well as map data from one flow to the other. It also serves as a
modularization technique because it can help you build reusable components.

Another important element—one that drives your flow—is an event. The user signals an event by
submitting the contents of a form to the URL that contains the flow id as a path and the flow execution
key as one of the parameters. Other parameters can be values of input components, such as text fields,
select boxes, check boxes, and so forth. Web Flow will use the flow id and execution key from the
submission to resume the flow execution, and it will parse the event from the submitted parameters.
Remember that, in our sample, the event is encapsulated in the name of the Submit button using this
format: _eventId_event. After parsing the event, Web Flow will use it to invoke the transition, which has
a matching on attribute for the event that we submitted. It will then be forwarded to the state indicated
by the to attribute of the transition.

A transition takes the flow from one state to another state of that same flow. Before you can go back
to a view, you must either transition to another view state (possibly going through other states) or
transition to an end state. As we will see, an end state has also an option to specify a view. This view is
rendered after the flow execution has terminated.

Different Web Flow Scopes

We already mentioned in the previous chapter that Web Flow adds a lot of different scopes into the
picture (five to be exact): conversation, flow, view, flash, and request (see the “Fine-Grained Scoping”
section in Chapter 10). Each of them has a well-defined lifecycle.

---

**Note** Every object you put on conversation, flow, view or flash scope should implement
java.io.Serializable (for the request scope, this is not required). This is the case because Web Flow stores
the state of these scopes between requests. The default implementation relies on Java serialization. Also note that
this is recursive; every object your object refers to should be serializable, as well. If you do not want to retain
certain objects, you can use the Java transient keyword on the field. This way, the field is exempted from being
serialized. When the flow state is restored, these fields will resolve to null.

---

Next, we will explain each of the five scopes in detail, from the shortest life cycle to the longest. Let’s
start with the request scope.

Request Scope

Figure 11-1 shows two HTTP requests entering the view state: HTTP request 1 and HTTP request 2.
While they are shown in the same figure, they should be interpreted sequentially. First, HTTP request 1
enters and gets processed, and the response is returned. Second, HTTP request 2 enters, and the cycle
repeats for this request. When the first request enters, request scope 1 is started. It is possible that an
event is triggered, and a transition is executed. Whatever is executed has access to that request scope.
When the processing is done, request scope 1 is destroyed, and the response for HTTP request 1
returned. When the next request (HTTP request 2) comes in, a new request scope (request scope 2) is created.