In this chapter, you will learn about portlet development using the Spring Portlet MVC framework, which is very similar to the Spring Web MVC framework because most of it is ported from Web MVC. You can often find their classes and interfaces with the same names but in different packages. Due to their similarity, I am not going to cover Spring Portlet MVC feature by feature, but rather focus on those of its portlet-specific features that are different from Web MVC. Before reading this chapter, please be sure that you have gone through Chapter 10, or have a basic understanding of Spring Web MVC.

A portal is a web site that collects information from different sources and presents it to users in a unified, centralized, and personalized way. This gives users a single access point to various information sources, such as applications and systems. In Java, a portal can use portlets to generate its contents. A portlet is a servlet-like web component technology that can process requests and generate responses dynamically. The content generated by a portlet is usually an HTML fragment that is aggregated into a portal page. Portlets need to be managed by a portlet container. The Java Portlet specification defines the contract between a portlet and a portlet container to ensure interoperability between different portal servers. In Spring 2.5, the Portlet MVC framework supports version 1.0 of this specification: JSR-168.

Upon finishing this chapter, you will be able to develop portlet applications using the Spring Portlet MVC framework and understand the differences between portlet and servlet development.

14-1. Developing a Simple Portlet with Spring Portlet MVC

Problem
You would like to develop a simple portlet with Spring Portlet MVC to learn the basic concepts and configurations of this framework.

Solution
The central component of Spring Portlet MVC is DispatcherPortlet. It mainly dispatches portlet requests to appropriate handlers that handle the requests. It acts as the front controller of Spring Portlet MVC, and every portlet request must go through it so that it can manage the entire request-handling process. When DispatcherPortlet receives a portlet request, it will
organize different components configured in the portlet application context to handle this request. Figure 14-1 shows the primary flow of request handling in Spring Portlet MVC.

When DispatcherPortlet receives a portlet request, it first looks for an appropriate handler to handle the request. DispatcherPortlet maps each request to a handler through one or more handler mapping beans. Once it has picked an appropriate handler, it will invoke this handler to handle the request. The most typical handler used in Spring Portlet MVC is a controller.

In portlets, there are two types of URLs: render URLs and action URLs. In most cases, there will be multiple portlets displaying in a page. When a user triggers a render URL, the portlet container will ask all the portlets in the same page to handle a render request to render its view, unless the view's content has been cached. A controller should return a ModelAndView object in response to a render request. However, when a user triggers an action URL in a portlet, the portlet container will first ask the target portlet to handle an action request. A controller needn't return anything for an action request. When the action request finishes, the portlet container will ask all the portlets in the same page, including the target portlet, to handle a render request to render its view.

After a controller has finished handling a render request, it returns a model and a view name, or sometimes a view object, to DispatcherPortlet. If a view name is returned, it will be resolved into a view object for rendering. DispatcherPortlet resolves a view name from one or more view resolver beans. Finally, DispatcherPortlet renders the view and passes in the model returned by the controller. Note that an action request doesn't need to render a view.