Dispatcher View

Context

System controls flow of execution and access to presentation processing, which is responsible for generating dynamic content.

Note

The Dispatcher View pattern, like the Service to Worker pattern, describes a common combination of other patterns from the catalog. Both of these macro patterns describe the combination of a controller and dispatcher with views and helpers. While describing this common structure, they emphasize related but different usage patterns.

Problem

The problem is a combination of the problems solved by the Front Controller and View Helper patterns in the presentation tier. There is no centralized component for managing access control, content retrieval or view management, and there is duplicate control code scattered throughout various views. Additionally, business logic and presentation formatting logic are intermingled within these views, making the system less flexible, less reusable, and generally less resilient to change.

Intermingling business logic with view processing also reduces modularity and provides a poor separation of roles among Web production and software development teams.

Forces

- Authentication and authorization checks are completed per request.
- Scriptlet code within views should be minimized.
- Business logic should be encapsulated in components other than the view.
- Control flow is relatively simple and is typically based on values encapsulated with the request.
- View management logic is limited in complexity.

Solution

Combine a controller and dispatcher with views and helpers to handle client requests and prepare a dynamic presentation as the response. Controllers do not delegate content retrieval to helpers, because these activities are deferred to the time of view processing. A dispatcher is responsible for view management and navigation and can be encapsulated either within a controller, a view, or a separate component.

Dispatcher View describes the combination of the Front Controller and View Helper patterns with a dispatcher component. While this pattern and the Service to Worker pattern describe a similar structure, the two patterns suggest a different division of labor among the components. The controller and the dispatcher typically have limited responsibilities, as compared to the Service to Worker pattern, since the upfront processing and view management logic are basic. Furthermore, if centralized control of the underlying resources is considered unnecessary, then the controller is removed and the dispatcher may be moved into a view.

Since the Service to Worker and Dispatcher View patterns represent a common combination of other patterns from the catalog, each warrants its own name to promote efficient communica-

tion among developers. Unlike the Service to Worker pattern, the Dispatcher View pattern suggests deferring content retrieval to the time of view processing.

In the Dispatcher View pattern, the dispatcher typically plays a limited to moderate role in view management. In the Service to Worker pattern, the dispatcher typically plays a moderate to large role in view management.

A limited role for the dispatcher occurs when no outside resources are utilized in order to choose the view. The information encapsulated in the request is sufficient to determine the view to dispatch the request. For example:

http://some.server.com/servlet/Controller?next=login.jsp

The sole responsibility of the dispatcher component in this case is to dispatch to the view `login.jsp`.

An example of the dispatcher playing a moderate role is the case where the client submits a request directly to a controller with a query parameter that describes an action to be completed:

http://some.server.com/servlet/Controller?action=login

The responsibility of the dispatcher component here is to translate the logical name `login` into the resource name of an appropriate view, such as `login.jsp`, and dispatch to that view. To accomplish this translation, the dispatcher may access resources such as an XML configuration file that specifies the appropriate view to display.

On the other hand, in the Service to Worker pattern, the dispatcher might be more sophisticated. The dispatcher may invoke a business service to determine the appropriate view to display.

The shared structure of these two patterns, as mentioned above, consists of a controller working with a dispatcher, views, and helpers.

Structure

Figure 1.1 shows the class diagram that represents the Dispatcher View pattern.

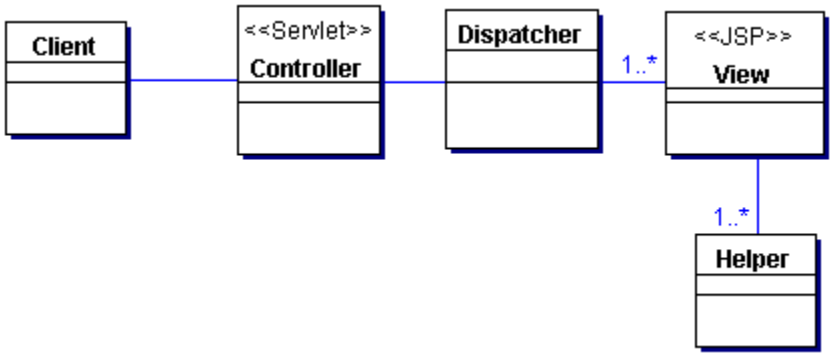

Figure 1.1 Dispatcher View class diagram.

Participants and Responsibilities

Figure 1.2 shows the Dispatcher View sequence pattern.

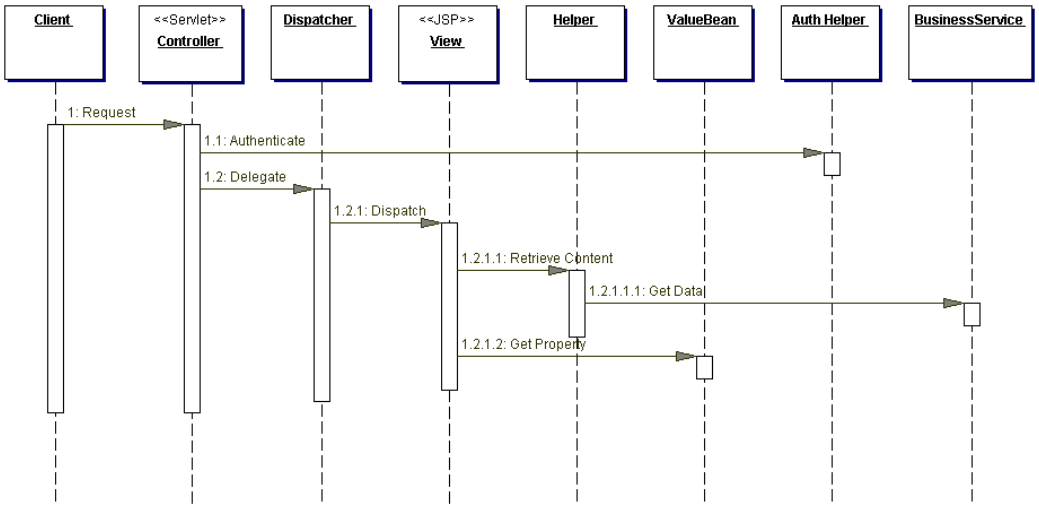

Figure 1.2 Dispatcher View sequence diagram.

While the controller responsibilities are limited to system services, such as authentication and authorization, it is often still beneficial to centralize these aspects of the system. Notice also that, unlike in Service to Worker, the dispatcher does not make invocations on a business service in order to complete its view management processing.

The dispatcher functionality may be encapsulated in its own component. At the same time, when the responsibilities of the dispatcher are limited, as described by this pattern, the dispatcher functionality is often folded into another component, such as the controller or the view In fact, the dispatcher functionality may even be completed by the container, in the case where there is no extra application-level logic necessary. An example is a view called `main.jsp` that is given the alias name `first`. The container will process the following request, translate the alias name to the physical resource name, and dispatch directly to that resource:

http://some.server.com/first --> /mywebapp/main.jsp

In this case, we are left with the View Helper pattern, with the request being handled directly by the view. Since the view is the initial contact point for handling a request, custom tag helpers are typically used in these cases to perform business processing or to delegate this processing to other components. Thus, the Dispatcher View pattern describes a continuum of related scenarios, moving from a scenario that is very structurally similar to Service to Worker to one that is similar to View Helper.

Controller

The controller is typically the initial contact point for handling a request. The controller manages authentication and authorization, and delegates to a dispatcher to do view management.

Dispatcher

A dispatcher is responsible for view management and navigation, managing the choice of the next view to present to the user and providing the mechanism for vectoring control to this resource.

A dispatcher can be encapsulated within a controller or can be a separate component working in coordination. The dispatcher can provide static dispatching to the view or may provide a more sophisticated dynamic dispatching mechanism.

View

A view represents and displays information to the client. The information that is used in a display is retrieved from a model. Helpers support views by encapsulating and adapting a model for use in a display.

Helper

A helper is responsible for helping a view or controller complete its processing. Thus, helpers have numerous responsibilities, including gathering data required by the view and storing this intermediate model, in which case the helper is sometimes referred to as a value bean. Additionally, helpers may adapt this data model for use by the view. Helpers can service requests for data from the view by simply providing access to the raw data or by formatting the data as Web content.

A view may work with any number of helpers, which are typically implemented as JavaBeans (JSP 1.0+) and custom tags (JSP 1.1+). Additionally, a helper may represent a Command object or a Delegate .

ValueBean

A value bean is another name for a helper that is responsible for holding intermediate model state for use by a view. A typical case, as shown in the sequence diagram in Figure 7.12, has the business service returning a value bean in response to a request. In this case, ValueBean fulfills the role of a Value Object.

BusinessService

The business service is a role that is fulfilled by the service the client is seeking to access. Typically, the business service is accessed via a business delegate. The business delegate's role is to provide control and protection for the business service.