View Helper

Context

The system creates presentation content, which requires processing of dynamic business data.

Problem

Presentation tier changes occur often and are difficult to develop and maintain when business data access logic and presentation formatting logic are interwoven. This makes the system less flexible, less reusable, and generally less resilient to change.

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

Forces

- Business data assimilation requirements are nontrivial.
- Embedding business logic in the view promotes a copy-and-paste type of reuse. This causes maintenance problems and bugs because a piece of logic is reused in the same or different view by simply duplicating it in the new location.
- It is desirable to promote a clean separation of labor by having different individuals fulfill the roles of software developer and Web production team member.
- One view is commonly used to respond to a particular business request.

Solution

A view contains formatting code, delegating its processing responsibilities to its helper classes, implemented as JavaBeans or custom tags. Helpers also store the view's intermediate data model and serve as business data adapters.

There are multiple strategies for implementing the view component. The JSP View Strategy suggests using a JSP as the view component. This is the preferred strategy, and it is the one most commonly used. The other principal strategy is the Servlet View Strategy, which utilizes a servlet as the view.

Encapsulating business logic in a helper instead of a view makes our application more modular and facilitates component reuse. Multiple clients, such as controllers and views, may leverage the same helper to retrieve and adapt similar model state for presentation in multiple ways. The only way to reuse logic embedded in a view is by copying and pasting it elsewhere. Furthermore, copy-and-paste duplication makes a system harder to maintain, since the same bug potentially needs to be corrected in multiple places.

A signal that one may need to apply this pattern to existing code is when scriptlet code dominates the JSP view. The overriding goal when applying this pattern, then, is the partitioning of business logic outside of the view. While some logic is best encapsulated within helper objects, other logic is better placed in a centralized component that sits in front of the views and the helpers—this might include logic that is common across multiple requests, such as authentication checks or logging services, for example. Refer to the "Intercepting Filter" on page 152 and "Front Controller" on page 172 for more information on these issues.

If a separate controller is not employed in the architecture, or is not used to handle all requests, then the view component becomes the initial contact point for handling some requests.

For certain requests, particularly those involving minimal processing, this scenario works fine. Typically, this situation occurs for pages that are based on static information, such as the first of a set of pages that will be served to a user to gather some information. Additionally, this scenario occurs in some cases when a mechanism is employed to create composite pages.

The View Helper pattern focuses on recommending ways to partition your application responsibilities. For related discussions about issues dealing with directing client requests directly to a view, please refer to the section on Dispatcher View.

Structure

Figure 1.1 is the class diagram representing the View Helper pattern.

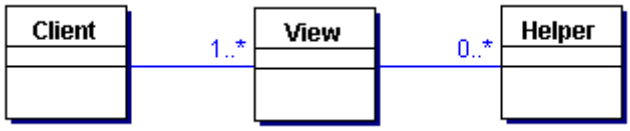


Figure 1.1 View Helper class diagram.

Participants and Responsibilities

Figure 1.2 shows the sequence diagram representing the View Helper pattern. A controller typically mediates between the client and the view. In some cases, though, a controller is not used and the view becomes the initial contact point for handling the request. (Also, see Dispatcher View pattern.)

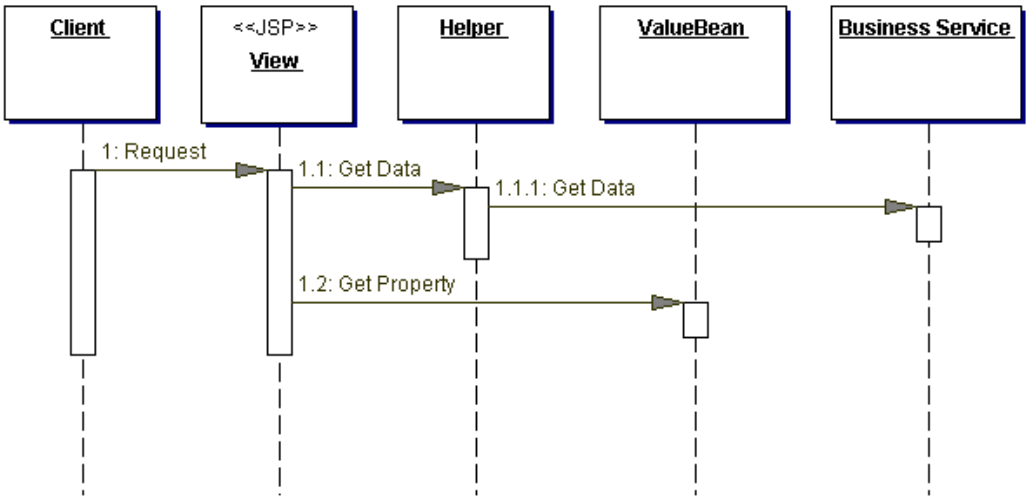
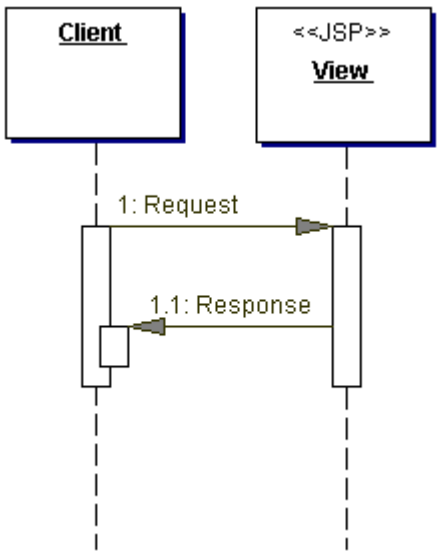


Figure 1.2 View Helper sequence diagram.

As noted in the class diagram, there may be no helpers associated with a view. In this simple case, the page may be entirely static or include very small amounts of inline scriptlet code. This scenario is described in the sequence diagram in Figure 7.13.

Figure 1.3 View Helper simple sequence diagram.



View

A view represents and displays information to the client. The information that is used in a dynamic 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, a delegate, or an XSL Transformer, which is used in combination with a stylesheet to adapt and convert the model into the appropriate form.

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 1.2, 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.