StrutsManualActionClasses

The goal of an Action class is to either implement a stateless _service_ like "Search", or to manage a stateful _business object_ like "Customer". An Action class handles a request and returns an ActionForward object, which represents a logical outcome of processing a request. An action mapping in {{struts-config.xml}} file associates ActionForward object with either another Action (see [action chaining|ActionChaining]) or with a presentation page. By not defining a specific target location in Java code, it is possible to separate logical outcome of an action from its visual representation. Struts is agnostic to presentation technology, so response can be generated using JSP file, Tile definition, Velocity template, XSLT stylesheet or other rendering engine. Base Action class handles all incoming requests with one callback method, {{execute()}}. Two overloaded versions of this method are available. Choosing one or another depends on your servlet environment: {{{public [ActionForward] execute(ActionMapping mapping, [ActionForm] form, [ServletRequest] request, [ServletResponse] response) throws Exception; public [ActionForward] execute(ActionMapping mapping, [ActionForm] form, [HttpServletRequest] request, [HttpServletResponse] response) throws Exception;}}} Since the majority of teams using the framework are focused on building web applications, most projects will only use the "HttpServletRequest" version. A non-HTTP {{execute()}} method has been provided for applications that are not specifically geared towards the HTTP protocol. h2. Using Action To Display A Web Page JSP is default presentation technology used when developing with Struts. JSP file creates dynamic web content by reading information from various Java objects stored in page, request, session or application scope. A standard practice to display a dynamic page in a Struts application is to use Action class "in front" of a JSP page (see [Model 2 web application architecture|http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web-tier5.html]). Action class creates needed beans, puts them in an appropriate context, and forwards control to a JSP page that reads information from these beans and displays it. Action class has access to all standard J2EE contexts except JSP-specific page context. As a concequence of Model 2 architecture, JSP pages are not directly accessible from browser when programming with Struts. Applications developed with Struts or with other similar web frameworks like WebWork or Stripes are often called _action-based_. This is different from so called _page-based_ frameworks like ASP.NET, where web page is accessed directly from browser. In ASP.NET a web page usually delegates events processing and page lifecycle-related tasks to a _code-behind_ class. In Microsoft parlance, Struts Action class can be called _code-in-front_. The following picture illustrates a "render page" use case implemented with Struts and ASP.NET. !basic_action_asp.gif! h2. Action And Setup/Submit Pattern The most common use case in an interactive web application is submitting of HTML form. A user expects that if input data is not valid then the form is redisplayed keeping information entered by the user, and displaying relevant error messages. The input/output process is traditionally implemented in Struts with setup/submit pattern: * _setup action_ (_pre-action_, _output action_, _render action_) prepares output data for display. It loads data from database, queues it into one or more arbitrary objects located in the request or session scope, then forwards to a view, usually a JSP page. * _submit action_ (_post-action_, _input action_, _accept action_, _event action_) processes input data from web form and redisplays the web form if errors has been found. If input does not contain errors, submit action updates application state and forwards to a success page. \\ ! setup_submit_simple.gif! A typical Setup Action will often implement the following logic in its {{execute}} method: * Make sure that a user is allowed to see the content of a web page that corresponds to the action. * Load needed data from database, set up a form bean or arbitrary request- or session-scoped objects. * Return an appropriate ActionForward object that identifies an appropriate presentation page. \\ A typical Submit Action will often implement the following logic in its {{execute}} method: * Validate the form bean properties as needed. If a problem is found, store the appropriate error message keys as a request (or session) attribute, and forward (or redirect) control to the setup action so that the errors can be corrected. * If input data is valid, perform the business task such as saving a row into a database. This can be done by logic code embedded within the Action class itself, but should generally be performed by calling an appropriate method of a business logic bean. * Update the server-side objects that will be used to create the next page of the user interface. These objects would typically be request scope or session scope beans, depending on how long you need to keep these items. * Return an appropriate ActionForward object that identifies the next web resource. \\ To ensure that the above pattern works correctly, a setup action should disable automatic form reset and population for forwarded requests (new feature of Struts 1.4). See tips on using setup/submit pattern and code sample (TODO link) The flip side of Struts flexibility is greater complexity of a most common use case of a web application. h2. Action As Event Dispatcher Event dispatcher handles a group of related _messages_ (_events_, _commands_). The difference between events and commands is subtle. Basically, event can be sent just "out there" and whoever is interested in that event handles it. Command is usually sent to a specific object that has to process the command. Messages often correspond to one business object, for example messages like Create, Retrieve, Update and Delete identify basic operations on persistent objects in a database-driven application. An event dispatcher defines methods that correspond to incoming messages; an Action that manages a persistent object will have methods {{create}}, {{retrieve}}, {{update}} and {{delete}}. Each method is triggered with a specific parameter sent in a request. Grouping related message handlers in one Action class reduces number of classes and often reduces number of mappings in {{struts-config.xml}} file. Struts defines several subclasses of Action class that perform event-dispatching, like DispatchAction, LookupDispatchAction, MappingDispatchAction, EventDispatchAction. These subclasses decode an event from incoming request and dispatch it to a corresponding event handler. The exact way of defining events depends on specific subclass. Note: to ensure that event handlers are always called, automatic validation should be turned off in {{struts-config.xml}} file. See tips on using event-dispatching action and code sample (TODO link) h2. Action As Web Resource Manager Many Struts users think in terms of simple actions and pages. It may be beneficial to think it terms of more generic web resources. A _web resource_ is a representation of a business object that an interactive application works with. For example, a Customer resource can be displayed in "View" and "Edit" modes, and can accept "New", "Edit", "Delete" and "Save" messages. It is possible to represent one logical entity with one event-dispatching Action. You can differentiate input and render phases either by request type (GET vs. POST) or by presence or absence of an event parameter in the request. Using one Action class to handle both input/render phases of a web resource brings the complexity of Struts web form management down to the level of ASP.NET while retaining the flexibility of a Model 2 framework. !web_resource_asp_simple.gif! See tips on using web resource manager and code sample (TODO link) h2. Action As Web Component Manager A web component is a web resource that is visually a part of a larger _composite page_. As such a web component does not need to navigate to a next location. Instead, a web component must either update itself on a composite page in place or reload the whole page after the component updated its state. In-place updating is facilitated using Javascript/XMLHttpRequest (Ajax mode), while full page reload is used if Javascript is turned off on a browser. !action_component_generic.gif! Struts 1.4 will allow creating web components using Struts Action class for event processing, JSP for presentation and an optional Javascript helper for in-place update in Ajax mode. See [Developing Web Components With Struts|StrutsManualActionWebComponent] on using web component manager and code samples.