

Design NetUIRequestProcess

NetUI Request Process

Steps in an Action Request

`PageFlowActionServlet.process()/AutoRegisterActionServlet.process():`

- Reinitialize reloadable class handler (bouncing a classloader if necessary). *This is for a "reloadable page flows without redeploy" feature that isn't fully implemented/supported currently.*
- Look up the appropriate Struts module based on the request module path (e.g., "/foo/bar" for "/foo/bar/someaction.do") and *register the Struts module if it isn't already registered*. Automatic module registration is where we look for the appropriate file under `/_pageflow` in the webapp classloader.

`PageFlowRequestProcessor.process():`

- Run request interceptors (see [RequestInterceptorContext](#)).
- Call back to the event reporter in the Servlet container adapter (see [ServletContainerAdapter](#)).
- Initialize the Controls context.
- Register the default URL rewriter

`PageFlowRequestProcessor.processInternal():`

- Create a request wrapper that contains request-scoped values that our runtime uses. This is faster than sticking everything into attributes on the request. From this point on, all request-scoped flags/values are accessed through this. See [PageFlowRequestWrapper](#).
- If this is a forwarded request, and if the "secure forwards" setting is enabled in `beehive-netui-config.xml`, allow the [ServletContainerAdapter](#) to do a security redirect to switch to https, if necessary.
- If the action was overridden by a request parameter (this is the mechanism for specifying the action on `<netui:button>`, forward to the appropriate action URI.
- If the request is for a page flow URI (.jpf), forward to the page flow's begin action.
- Get the FlowController for this request (page flow or shared flow), and cache it in the request.
- Look up or create any shared flows that are needed for the current request (based on the current page flow and any global shared flows defined in `beehive-netui-config.xml`. `ImplicitObjectUtil.loadSharedFlow` puts a Map of shared flows (name -> instance) into the request at the right spot.
- Remove any current JavaServer Faces backing bean. We're in action processing now, and the user is no longer interacting with a JSF page.
- Set up all the implicit objects for databinding (pageFlow, etc.), through `ImplicitObjectUtil.loadImplicitObjects`.

Struts `RequestProcessor.process()` (note that I'm only listing the bits of this that we override in `PageFlowRequestProcessor`):

- `processNoCache`: In addition to the default Struts behavior of setting no-cache headers if it's configured in the Struts module, we set the headers based on global settings in `beehive-netui-config.xml`. We also put a flag in the request that our *page filter* uses to set no-cache headers if we forward to a NetUI page.
- `processMapping`: This is the step that looks up which action to run, based on the request path. Here, we do a number of things:
 - If this is a request for a shared flow action mapping (*shared-flow-name.action-name*), look up the action mapping in the appropriate shared flow.
 - If there is a form bean that was forwarded from another action, or returned from a nested page flow, look up a form-bean-specific action mapping. This is part of support for overloaded actions (actions with the same name but that accept different form bean types).
 - Look up the action mapping in the current Struts module (for the current FlowController).
 - If the action still isn't found, try it in the (deprecated) Global.app module.
- `processRoles`: If the user isn't in the correct role (as defined in annotations), throw an exception. [NotLoggedInException](#) if the user isn't logged in, or [UnfulfilledRolesException](#) if the logged-in user isn't in the correct role. This is different than default Struts behavior, which is to send an error on the response. The exception that we throw here can be handled by exception handling annotations in the page flow.
- `processActionForm`: We deal with two things here beyond default Struts behavior:
 - If there is a form bean that was forwarded from another action (or returned from a nested page flow), we use that instead of creating a new instance.
 - If the current action has the `useFormBean` attribute set in its annotation, then we get/set the page flow controller's member variable which was named.
- `processPopulate`: This is where all of our databinding (writing values based on request parameters) takes place. Obviously a lot more here than there is in Struts.
- `processActionCreate`: We simply create a [FlowControllerAction](#) that delegates to this request's FlowController, which we've already looked up and cached. As mentioned above, the FlowController may be a page flow or a shared flow.
- `processActionPerform`:
 - Run before-action methods on any registered action interceptors. See [ActionInterceptor](#).
 - Execute the action.
 - Run after-action methods on any registered action interceptors.
- `processForwardConfig`: This just contains our custom logic for forwarding/redirecting to a URI.

Back in `PageFlowRequestProcessor.process()`:

- If this is the end of a series of forwarded requests, apply changes in the [StorageHandler](#). See [DeferredSessionStorageHandler](#) for more info.
- Uninitialize the Controls context.
- Call back to the event reporter in the Servlet container adapter (see [ServletContainerAdapter](#)).
- Execute any post-intercept code in registered request interceptors.

Steps in a Page Request

`PageFlowPageFilter.doFilter()`:

- If we decided during an earlier action request that the response shouldn't be cached, set the no-cache headers.
- Call back to the event reporter in the Servlet container adapter (see [ServletContainerAdapter](#)).
- Initialize the Controls context.
- Register the default URL rewriter
- Ensure that the right Struts module is registered and selected in the request, for use by the tags.
- Look up or create any shared flows that are needed for the current request (based on the current page flow and any global shared flows defined in `beehive-netui-config.xml`. `ImplicitObjectUtil.loadSharedFlow` puts a Map of shared flows (name -> instance) into the request at the right spot.
- Make sure the current page flow ([PageFlowController](#)) is set up for the request. Unless an earlier action request specified that we should stay in the current page flow's module, we'll look up the appropriate page flow based on the request URL.

`PageFlowPageFilter.runPage()`:

- Check to see if there are too many concurrent requests to the same page flow. This prevents an attack that takes advantage of the fact that we synchronize requests to the same page flow. If there are too many concurrent requests, send an error on the response; otherwise, increment the request count.
- Synchronize on the current page flow controller instance.
- Set per-request state on the controller, like `current-request` and `current-response`. This is so these methods can be used if the page databinds to getter methods in the controller.
- Call back to the current page flow controller (`beforePage` – this is internal, for now).
- Run the page
- Clear per-request state on the controller.
- Decrement the request count on the controller.

Back in `PageFlowPageFilter.doFilter()`:

- Uninitialize the Controls context.
- Call back to the event reporter in the Servlet container adapter (see [ServletContainerAdapter](#)).
- If this is the end of a series of forwarded requests, apply changes in the [StorageHandler](#). See [DeferredSessionStorageHandler](#) for more info.

Future Directions

These two processes (Action Request and Page Request) are ideal candidates for something like Chain, which is used by Struts 1.3. Each step could be turned into a chain command, and the process could be configured in Chain's XML config. As you can see, many of the steps are shared by both Action and Page requests.