Design NetUIHandlers

NetUI Handler Layer

Introduction

(Note, this is really more user documentation than design doc. At some point it can be morphed into actual Beehive doc.)

Customization of NetUI framework behavior is done through *Handlers*. A Handler is simply an implementation of an interface that provides some sort of framework behavior. An example is LoginHandler, which provides methods for logging in a user, logging out a user, and testing whether a user is logged in and which roles a user is in. A developer may want to override the default framework login behavior to use a user database, instead of using the default Servlet login behavior in NetUI.

Handler Types

There are currently six Handlers in NetUI:

- ActionForwardHandler: Deals with processing of Struts ActionForwards.
- ExceptionsHandler: Deals with all exception handling. Common functionality to override here is unwrapException, which unwraps through wrapper exception types so that Page Flow exception-handling annotations can stick with the underlying types as the ones to handle.
- ForwardRedirectHandler: Controls basic Servlet forwarding and redirecting.
- LoginHandler: Controls logging in a user, logging out a user, and testing whether a user is logged in and which roles a user is in.
- StorageHandler: Deals with getting/setting session-scoped attributes, and ensuring that attributes get failed over in a cluster.
- ReloadableClassHandler: Deals with loading and reloading of all classes used by Page Flow. Part of support for "reloadable-without-redeploy" page flows, which is a feature that's not fully supported.

Configuring Handlers

Handlers are set up so that any number of a particular type of Handler can be configured, with one delegating to another. Each Handler can get a reference to the previous handler in the list, and can delegate as needed. **This is much like Java inheritance, but each Handler doesn't have to know about all the others at compile time.**

Handlers are configured in `beehive-netui-config.xml`, in the `pageflow-handlers` section. Here is an example of a list of configured Handlers. In the case of `login-handler`, where there are two, we can assume that each one delegates to the one before it unless it is overriding a particular method.

```
<pageflow-handlers>
    <forward-redirect-handler>
        <handler-class>example.MyRedirector</handler-class>
    </forward-redirect-handler>
    <login-handler>
        <handler-class>example.OverrideLoginHandler1</handler-class>
    </login-handler>
    <login-handler>
        <handler-class>example.OverrideLoginHandler2</handler-class>
    </login-handler>
</pageflow-handlers>
```

Handlers can be configured with custom properties that are available at runtime, e.g.,

```
    <login-handler>
        <handler-class>example.OverrideLoginHandler3</handler-class>
        <custom-property>
            <name>prop1</name>
            <value>value1</value>
        </custom-property>
        <custom-property>
            <name>prop2</name>
            <value>value2</value>
        </custom-property>
    </login-handler>
```

Implementing Handlers

To implement a Handler, two things are normally done:

- extend BaseHandler. This gives access to the ServletContext, and also provides a reference to the previously-registered Handler of the same type (the one that comes before the current one in the list under `<pageflow-handlers>` in beehive-netui-config.xml).
- implement the desired Handler type, e.g., LoginHandler.

Here's an example of an `ExceptionsHandler`, which overrides the base behavior for unwrapping exceptions, but otherwise just delegates:

```
    public static class OverrideExceptionsHandler
        extends BaseHandler
        implements ExceptionsHandler
    {
        public ActionForward handleException(FlowControllerHandlerContext context, Throwable ex, ActionMapping
actionMapping, ActionForm form)
            throws IOException, ServletException
        {
            return getPreviousExceptionsHandler().handleException(context, ex, actionMapping, form);
        }

        public Throwable unwrapException(FlowControllerHandlerContext context, Throwable ex)
        {
            if (ex instanceof MyWrapperException()) {
                ex = ((MyWrapperException) ex).getCause();
            }
            return getPreviousExceptionsHandler().unwrapException(context, ex);
        }

        public void exposeException(FlowControllerHandlerContext context, Throwable ex, ActionMapping
actionMapping)
        {
            getPreviousExceptionsHandler().exposeException(context, ex, actionMapping);
        }

        public boolean eatUnhandledException(FlowControllerHandlerContext context, Throwable ex)
        {
            return getPreviousExceptionsHandler().eatUnhandledException(context, ex);
        }

        protected ExceptionsHandler getPreviousExceptionsHandler()
        {
            return (ExceptionsHandler) super.getPreviousHandler();
        }
    }
```

Getting Handlers at Runtime

To get a Handler, just call Handlers`.get()` to retrieve a Handlers object, then `getLoginHandler()`, `getExceptionsHandler`, etc.