Design PortletScoping

Portlet Scoping

Introduction

As underlying support for running NetUI applications within portlets (originally within BEA's proprietary Portal), the idea of *scoping* is present throughout the NetUI framework. There are two kinds of scoping:

- *Session Scoping*: Simple. It means that things like page flow instances are stored in the session under "scoped" names, which are the usual internal attribute names with a Scope ID (portlet ID) prefix.
- *Request Scoping*: More complicated underneath, but simple in concept: prefix all request parameter and attribute names with the Scope ID (portlet ID).

Session Scoping

Details

All getting/setting of session-scoped attributes for page flow, page flow nesting stack, shared flow, and JSF backing bean instances uses `ScopedServletUtils.getScopedSessionAttrName()` to construct the actual session attribute name. By default, this API returns the attribute name unmodified. If either of the following two conditions are true, then the Scope ID is prefixed to the attribute name:

- The current request is a `ScopedRequest`. `ScopedRequest` is a wrapper over `HttpServletRequest` that we use when running inside a portal (more details below).
- A special request parameter (`ScopedServletUtils.SCOPE_ID_PARAM`) exists. This is *not inherently part of portal support.* It is used to support multiple active page flows in the session, usually for multiple browser frames/windows. It's important to note that the scoping mechanism is exactly the same as what it is for `ScopedRequest`/Portal, so a browser window using this mechanism could talk to the page flow from a *portlet* with the same Scope ID... but in the end it can be used on its own. The `targetScope` attribute on NetUI tags like `<netui:anchor>` is a convenience for setting this parameter.

Issues and Future Directions

If NetUI changes to use a NetUI-controlled *context object* for things like request/session access, then this context object can be smart about whether to create scoped attribute names. It would eliminate the need for calling `ScopedServletUtils.getScopedSessionAttrName()` whenever setting a managed session attribute.

Request Scoping

URL Rewriting

A portal framework can register a *URL rewriter* that will prefix request parameters with the current Scope ID in any page rendered using NetUI tags. This makes the names unique, and targeted to a particular portlet. Some examples (assuming a portlet ID of "portletA":

- `<input type="text" name="portletAfoo">` instead of `<input type="text" name="foo">`
- `<a href="/myApp/someurl?portletAbar=someValue` instead of `<a href="/myApp/someurl?bar=someValue`

The URL rewriter may rewrite the URL in other ways, but this is its main function as it relates to scoping.

ScopedRequest

A portal framework can create a `ScopedRequest` wrapper around a basic `HttpServletRequest` in order to decode scoped requests (normally requests that come from a page rewritten with a URL rewriter, above). The `ScopedRequest` does two main things:

- keeps track of a Scope ID (portlet ID)
- *filters* request parameters and attributes to return only ones that have the desired Scope ID. In the URL rewriter examples above, a `ScopedRequest` with Scope ID of "portletA" would see request parameters "foo" and "bar", respectively.

In general, a `ScopedRequest` would be created/initialized by a portal framework in framework code (or even in a Servlet Filter) before NetUI ever sees the request.

Issues and Future Directions

A `URLRewriter` should be responsible for **decoding** scoped parameter names, in addition to its current responsibility for rewriting parameter names to be scoped. Currently, our `ScopedRequest` (and the rest of the scoping package) assumes that scoping is done by prefixing the Scope ID to the name. A portal URL rewriter currently has to know this implicitly.

A `ScopedRequest` obtained through `ScopedServletUtils.getScopedRequest()` can be configured to "see" a request attribute on the outer request if it's not present in the scoped request. NetUI may currently break when this is turned on (something to do with URL rewriters). We should make sure to work in this situation.

A portal framework is in charge of persisting scoped request attributes as necessary for use during "refresh" requests (requests that involve user interaction with a different portlet than the one being refreshed). In the case where the NetUI framework knows an attribute *should* be persisted, it should use some API for marking a request attribute as "persistable". Something like `ScopedServletUtils.setPersistableAttribute()` and `ScopedServletUtils.markPersistableAttribute()` (the latter would be to mark an attribute that's already been set, like some of the ones set by Struts).

Related to the above, there are some attributes that should be persisted, but which are large. An example is a Struts `ModuleConfig`, which can be reconstructed based on the module path. NetUI could offer a `ReconstructibleAttribute` abstract base class that `ScopedRequest` would be aware of:

```
public abstract class ReconstructibleAttribute
{
    private transient Object attribute;
    private Object reconstructionKey;

    public ReconstructibleAttribute(Object attribute, Object reconstructionKey) {
        this.attribute = attribute;
        this.reconstructionKey = reconstructionKey;
    }

    public void dropAttribute() {
        attribute = null;
    }

    public abstract void reconstructAttribute(ServletRequest request, ServletContext servletContext);
}
```

Also, a `PeristableAttribute` for attributes that can be persisted as-is:

```
public class PersistableAttribute extends ReconstructibleAttribute
{
    public PersistableAttribute(Object attribute) {
        super(attribute, attribute);
    }

    public void reconstructAttribute(ServletRequest request, ServletContext servletContext) {
        attrible = key;
    }
}
```

Calling `ScopedServletUtils.setPersistableAttribute` would set a `ReconstructibleAttribute` in the `ScopedRequest`. Calling `getAttribute()` on `ScopedRequest` would get the *value* of a `ReconstructibleAttribute` (reconstructing as necessary). When persisting attributes, a portal framework could call `dropAttribute()` on anything that derives from `ReconstructibleAttribute` in the map returned by `ScopedRequest.getAttributeMap()`.

Response Scoping

NetUI also includes a `ScopedResponse` wrapper that a portal framework would use to wrap the actual response. The default implementation doesn't do much beyond keeping track of whether a redirect happened, and of errors/status-messages/cookies.

Public APIs Related to Scoping

The most commonly-used APIs are in `ScopedServletUtils`:
`getOuterRequest`: Gets the outer request, which is the one that the `ScopedRequest` is wrapping. If there is no `ScopedRequest`, just returns the given request.

`unwrapRequest`: Badly named. Unwraps until it finds a `ScopedRequest`, which it returns. If there is none, returns null.

`getScopedRequest`: Used by a portal framework to create or look up a cached `ScopedRequest`, based on the original request and a Scope ID.

`getScopedResponse`: Used by a portal framework to create or look up a cached `ScopedResponse`, based on the original response and a `ScopedRequest`.

Additionally, a portal framework will want to use the `URLRewriterService` API:
`registerURLRewriter`: Register a `URLRewriter` with the request to prefix request parameters with the current Scope ID.

`unregisterURLRewriter`: Unregister the URL rewriter from the request.