BrowserBackAndSecurity

Managing the Browser Back Button

Two commonly asked questions on the Struts Users mailing list are how to prevent the user from seeing stale data when they hit the browser's 'back' button, and how to prevent them from seeing a secured page by hitting the back button after logging out. It turns out that these issues are related, and the solution to the latter builds on the solution to the former.

Browser Caching Behaviour

When you ask your browser to display a web page it has several options for how to proceed, depending on whether the page has been requested before, whether it has a copy cached locally, and what caching policies affect that page. Assuming the browser has displayed a page at least once, the options are (roughly speaking):

- display the page from the browser cache
- ask the server if there is a newer version of the page than what's in the cache. If not, display the cached copy; otherwise, fetch the updated copy
- fetch a fresh copy every time

Browsers can usually be configured to force the use of any of these strategies all the time. Typically, however, the behaviour depends on hints the browser receives from the server whenever it requests a page. These hints tell the browser whether it is allowed to cache the page at all and, if so, under what circumstances and for how long. By default, a browser will do its best to cache pages and avoid fetching fresh copies, so it can display them more quickly when asked to do so.

Avoiding Stale Date: Cache Control

To prevent the user from seeing stale data when they hit the back button, you need to tell their browser not to cache the page. You do this by including cache control 'hints' – special HTTP headers – in the response to each request. There are a number of different 'hints' you can supply, and different browsers (and HTTP proxies) respect different hints, so you need to supply several to achieve the desired result in all circumstances. Specifically, you need to set the `Pragma`, `Cache-Control`, and `Expires` headers, as follows:

```
Pragma: nocache
Cache-Control: no-cache, must-revalidate, no-store
```

You can do this every time in your actions or JSPs, but that quickly gets tedious. Fortunately, Struts provides a way to do this automatically, by setting `nocache="true"` on the `controller` element in `struts-config.xml`:

```
<controller nocache="true" processorClass="...
```

With this set, responses from your Struts application will include the cache-control headers and the browser will never cache them. As a result, when the user hits the 'back' button, the browser will always request a fresh copy of the page and will never be presented with a cached, stale copy.

Data Caching vs. Browser Session History

Technically, there is a difference between caching of resource data and storing resource location in useragent's page history.

- Clicking on a link - only data cache is checked
- Moving back or forward - data cache and/or page history may be checked depending on browser vendor
- Forcing page reload - cache is not checked, page is reloaded

According to section 13.13 of HTTP 1.1 specification a user agent should not reload data from resource while navigating page history. This is exactly the reason why Opera does not reload a page when you click Back even if the page is marked as non-cachable:

```
RFC2616, 13.13 History Lists
----------------------------
History mechanisms and caches are different. In particular history mechanisms SHOULD NOT try to show a
semantically transparent view of the current state of a resource. Rather, a history mechanism is meant to show
exactly what the user saw at the time when the resource was retrieved.
```

Other browsers behave differently. For example, MSIE reloads a page in both SSL and non-SSL mode if response contains "no-cache" cache-control header. Mozilla reloads a "no-cache" page only in SSL mode, for non-secure mode it requires "no-store" header.

One might say that Opera provides cleaner separation between data caching and location history. On the other hand, for many stateful web applications this behavior ruins the idea of page<->data synchronization at any given time. Mozilla seems to be a good compromize between MSIE, that basically puts equal sign between data caching and location history, and Opera.

Enforcing Page Security

The problem with with users accessing secured pages by hitting the back button after logging out is partially solved by implementing cache control as described above, but that is only part of the solution. With cache control in place, the browser will request a fresh copy of the secured page but it is up to the application to determine that the user is no longer logged in and refuse to display the requested page.

There are many ways of securing pages in a web application. Approaches include Container-managed security, using a Servlet Filter to pre-process every request, or implementing security checks in your individual actions (among others). We wont cover the details here, just note that you need some form of **per-request** access control in place. Every time a user requests a page, a check needs to be made (by the Servlet container or by your application) that the user is logged in and has permission to see the page. With that in place, together with cache control, you can be assured that when the user logs out they will not be able to return to a secured page using the back button.

Typically, you will base your access control on the user session. In that case, logging the user out is simply a matter of invalidating the session (calling `Session.invalidate()`). If your access control mechanism uses something other than the session – such as a cookie – to track a user's authenticaion status, you will need to manage that appropriately.

For a good introduction to implementing security and access control in a Struts-based web application, see the following resources:

Links to tutorials / articles

- http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.9
- http://www.web-caching.com/