Sakai Sessions

March 23, 2005

Please direct questions or comments about this document to:
Glenn R. Golden, ggolden@umich.edu

Sakai differs from other single web applications in that it is deployed over a number of separate web applications in a Servlet container. While the Servlet API defines capabilities for user session information within each web application, it does not define a user session capability across all web applications.

Sakai's Session API provides this Sakai-wide session capability. Modeled on the HttpSession of the Servlet 2.4 Spec, a Sakai session is scoped at the end user level and shared among all Sakai applications, each in a different webapps, and the Sakai framework.

Sakai differs from other web applications, but is similar to portal applications, in that our Servlets are usually not mapped directly into URL space, but instead enable the processing of Sakai Tools, which can exist multiple "places" in a Sakai installation. Each tool placement has a separate configuration and end user interaction state.

Sakai's Session API supports ToolSessions, one for each tool placement, living within the Sakai-wide session for each end user. A ToolSession is a subset of a full Session, used to store attributes that capture the end user interaction state.

The Sakai Session API can be used to supply a Sakai managed HttpSession for Servlets, instead of the HttpSession managed by the Servlet container. This appears as a real HttpSession to the Servlet.

The interfaces in the Session API (Session, SessionBindingListener, SessionBindingEvent) are closely modeled on the similar interfaces in the Servlet API (HttpSession, HttpSessionBindingListener, HttpSessionBindingEvent), but do not directly extend these Servlet APIs. We make them similar because they do similar things, and so that developers can leverage their understanding of the Servlet API when working with Sakai. We keep them from directly extending to keep the Sakai Session API code independent of the Servlet API.

Sakai Sessions

A Sakai session is just like an HttpSession from the Servlet 2.4 API, without the deprecated methods.

Like the HttpSession, a Sakai session can hold attributes, and expires after a period of time unless refreshed by a client request to any Sakai web application. Values used in

session attributes can implement the SessionBindingListener interface and be informed when they are bound or unbound from a Session.

Sakai sessions also store the uuid and enterprise id of the authenticated end user as a first-class property.

There is one Sakai Session for each end user, available to all Sakai applications.

Sakai Tool Session

A tool session is a subset of the full Session interface, providing a place to hold attributes that are scoped to a single user's interaction with a single tool placement. Values used as attributes in a tool session can, like those in a Session, be notified when they are bound or unbound.

Session Manager

The SessionManager manages Sakai sessions and tool sessions. The APIs for sessions are defined in the Sakai `kernel` module, in the `session` project.

The SessionManager is responsible for the creation of, access to, and maintenance of Sessions.

Current Session

The Session API supports the idea of a "current" session. This is the Session object associated with the end user behind the current request processing thread. Any Sakai code can ask the SessionManager for the current session via the `getCurrentSession()` method.

The API also supports the idea of a "current" tool session. This is the ToolSession object associated with the end user and the particular tool placement of the current request. The SessionManager's `getCurrentSession()` provides this information.

The current session capabilities are implemented using the Sakai CurrentManager, and is aided by the Sakai Request Filter.

Sessions in a Web Server Environment

When Sakai is running in a web server environment, the Servlet container manages each webapp's HttpSession, usually using a cookie called `JSESSIONID`, written by the container to the end user's browser. There is a separate cookie for each web application. The browse sends the appropriate cookie with each request to each webapp. The Servlet API has no provision to manage a global session.

The Sakai Session is managed in a similar way. Sakai sends a single cookie called `SakaiSessionId` to the end user's browser, target at the root of the Sakai web server. The browser sends this one cookie back to any request to Sakai. Each webapp installed in Sakai sees the cookie and has the end user's session id available when processing

requests. This cookie based tracking of the Session is handled in the request filter. The filter also makes available the current Session and ToolSession, and places these in request attributes. See the Sakai Request Filter document for more details.

Use of the Session

The Sakai Framework uses the session to store the authenticated end user id. It may store other Sakai-wide per-user information there as well.

Sakai applications can also use the session to store per-user session information. Information used by many different applications or placements of tools can be stored here. Take care to establish a namespace that separates this information from all other possible users of the Sakai session.

The Sakai framework uses attribute names beginning with "`sakai.`" for framework-scoped information.

Information related to the application's end user interaction is better stored in the ToolSession for this particular placement of the application's tool.

Sakai Managed HttpSession

For Servlets and other Servlet technologies working as a Sakai tool, which already use an HttpSession, Sakai can manage the HttpSession. A Sakai managed http session is really the Sakai-side session; there is just one for all web applications. This is wider in scope than a normal Servlet-container managed http session, which is scoped to have one per web application.

The advantage of using a Sakai Managed HttpSession is that you do not have to worry about how your Servlet is invoked. Tomcat will give a different session for invocations that are direct by URL than those that are via a request dispatcher from another web application. Sakai may use either or both of these methods to invoke tool code. Using a Sakai managed http session guarantees session consistency no matter how the Servlet is invoked.

Another problem occurs when new HttpSessions are created – at that point, the Servlet container needs to establish a JSESSIONID cookie. Cookies go into the response header, and if the response has been committed already, cannot be written. Servlets that are used in various positions in Servlet "chains", i.e. that are invoke by other Servlets using a request dispatcher, cannot know that they are free to write headers, since the prior processing for this response may have committed the response. The Sakai managed HttpSession already exists before a Servlet is invoked, so we avoid the problem.

There are some limitations on using a Sakai managed http session. First, there is just one per user, so the namespace of attributes must be carefully coordinate between all the web applications. Usually you need to only carefully coordinate it between the different components of a single web application.

Session API Implementation

Sakai ships with a standard component that implements the Session API. This is in the `kernel` module's `session-component` project. It is deployed as part of the `kernel` module's `kernel-components` project. It is unlikely that any Sakai installation would want to swap this out for another implementation, but this is possible.