Sakai Request Processing

July 20, 2005

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

The Sakai Application Framework kernel includes support for http request processing in the form of a Servlet Filter. This filter is installed in all Sakai web applications in front of all Sakai request paths and tool Servlets. The filter pre- and post- processes the request to achieve the special request handling needed in Sakai. This includes:

- Sakai Session management

- HttpSession management

- Remote User management

- Tool Placement URL encoding and detection and ToolSession management

- ThreadLocalManager setup and cleanup

- Character encoding management

- File Upload management.

All of the features of the filter are configurable and may be disabled if desired in certain uses of the filter.

Using the Filter

Every Sakai tool registered in a web.xml file of a web application must have the filter installed in order to take full advantage of Sakai features and call on Sakai API.

The Filter is defined with some XML in the web.xml file, like this:

```
    <filter>
        <filter-name>sakai.request</filter-name>
        <filter-class>
            org.sakaiproject.util.RequestFilter
        </filter-class>
</filter>
```

If you want to control the filter options with other than default values, specify them with init-param entries like this (Note: these examples are all default values):

```
<filter>
        <filter-name>sakai.request</filter-name>
        <filter-class>
            org.sakaiproject.util.RequestFilter
        </filter-class>
        <init-param>
            <param-name>http.session</param-name>
            <param-value>tool</param-value>
        </init-param>
        <init-param>
            <param-name>remote.user</param-name>
            <param-value>true</param-value>
        </init-param>
        <init-param>
            <param-name>tool.placement</param-name>
            <param-value>true</param-value>
        </init-param>
    </filter>
```

You need just one filter definition if you are going to use it on one or more Servlets in the web application, if they all use the same options.

You then need to map the filter to have it applied to the Servlets:

```
    <filter-mapping>
        <filter-name>sakai.request</filter-name>
        <servlet-name>sakai.sample</servlet-name>
        <dispatcher>REQUEST</dispatcher>
        <dispatcher>FORWARD</dispatcher>
        <dispatcher>INCLUDE</dispatcher>
    </filter-mapping>
```

This maps the filter to the Servlet by name (`sakai.sample` in this example). Put in a filter mapping for each Servlet in the web application that is a Sakai tool or uses Sakai framework features. Map the filter to the Servlet name.

Multiple Filter Invocations

It is possible that the filter could be invoked many times while processing a single request. We need to register it for REQUEST as well as FORWARD and INCLUDE so that the filer is used for direct URL requests to the Servlet, and for request dispatcher invocations. Many Servlets may be involved in the processing of a single Sakai request; a Portal or Navigator, a series of Tools, and possibly Helper Tools. The initial Servlet (usually the Portal / Navigator) is invoked by REQUEST, the rest are invoked by FORWARD or INCLUDE via a request dispatcher. Each of these invocations will invoke the filter.

The request filter will only do its tasks once per request. The filter may be invoked more than once, but each of the various tasks that the filter manages will only happen once per request. The filter internally tracks session management, request character encoding, and file upload parsing so that each of these tasks will happen at most once on a single request. This is implemented internally by a marker attribute on the request.

Session Management

The filter is responsible for assuring that the Sakai Session is available. It does this by writing a cookie, `SakaiSessionId`, to the end user's browser, containing a session id.

When each request arrives, the filter looks for the cookie, and finds the session with the id that is stored in the cookie. If the cookie or session is missing, a new session is created for the user.

The session is set as the "current" session for this request thread. This session is also placed in a request attribute, `sakai.session`, for later use by the Sakai tool processing the request

If the cookie was missing or a new session was created, the cookie is written to the browser. This is taken care of before the filter sends the request on for further processing, to assure that the new cookie can be written and that the request is not yet committed.

Every request that goes through the filter ends up with a Sakai session created for the end user.

HttpSession Management

Sakai offers the ability to manage the HttpSession for the Servlet. This is enabled with the `http.session` init parameter. If enabled, the HttpSession for the Servlet will exist, as if another Servlet in the web application created it. It looks and acts like a real HttpSession. It happens to be the Sakai Session, though.

A Sakai session can have three scopes, controlled with different settings for http.session:

- container
- sakai
- context
- tool

Setting http.session to "container" disables Sakai's special session management. Setting it to "sakai" gives the request access to a single session shared among all of Sakai. Setting it to "context" gives a session shared by any other tool in the same servlet context, or set to the same context name. Setting it to "tool" (the default) gives the request access to a session that is scoped only to this tool.

Letting Sakai manage a Servlet's HttpSession solves some problems. The Servlet gets the same session (per end user) when it is invoked by URL or by request dispatcher. For Tomcat managed HttpSessions, this is not always the case. The Servlet can also ask for the session at any time while processing the request, and the Servlet can be used anywhere in the request's filter / Servlet chain (made by filters and Servlets using request dispatchers), and still ask for a session. With Tomcat managed HttpSessions, once any element in the chain causes the request to commit, new sessions cannot be created, because the cookie cannot be written at that point.

Sakai managed sessions differ from Tomcat or Servlet-container managed sessions. They can have a scope that might covers all web applications, not just one (when using the "sakai" scope). This means you have to be more careful with the attribute names bound to the session.

This is a completely optional feature. Sakai does not require it for proper functioning.

To share the HTTP session across multiple tools, you can use the "context" setting for `http.session`. You can then either place the tools in the same servlet context (the servlet context name is used as the context string to identify the session to use), or configure the filter with the `context` init-param set to a common string. This way you can have a session shared between tools in different webapps.

Note: tools that invoke other tools as helper tools naturally share the invoker's HTTP session, and do not need to have special session sharing enabled in the filter.

Remote User

The filter will find the authenticated user's enterprise id from the Sakai Session, and set this as the Request object's `REMOTE_USER` value. This can be disabled with the `remote.user` init parameter to the filter. This is a simple way for a Servlet to participate in "single sign-on" with other Sakai applications; it can get the user EID in this standard way.

Tool Placement / Tool Session

The filter enables the distinguishing of tool placement in URL requests to tools, in some situations. Tool placement allows a single tool to be used as if it were many instances of the single tool, each with a different configuration and end user interaction state. Each of these instances is allocated a unique placement id.

When the tool Servlet writes a URL into the response that is a URL back to the Servlet, and there's a current tool placement in effect, the filter will add the placement id to the URL, as a request parameter.

When the filter processes an incoming request, it detects the presence of the placement id, and makes it available for further processing. It places it in a request attribute called `sakai.tool.placement` if found. If this attribute is left in place, the filter will encode the value into URLs back to the Servlet.

In order for the URL encoding to work, the Servlet must call the response object's `encodeURL()` methods, as called for in the Servlet spec, whenever outputting a URL.

Once the filter detects a placement id, it uses it to find the ToolSession object of the Sakai Session, created as needed. The filter sets this as the "current" tool session for the request. It also sets the ToolSession object into the request attributes as `sakai.tool.session`.

Tool placement processing can be configured using the `tool.placement` init parameter for the filter.

Any tool invoked through a Sakai navigator or portal need not worry about this, as the navigator / portal takes responsibility for tool placement. This can be used in front of a not-very-Sakai Servlet to help it be usable in Sakai.

ThreadLocalManager

The Sakai ThreadLocalManager makes available information bound to the current request processing thread, using the Java "thread local" paradigm. The filter is responsible for final cleanup of any bound information when the request has completed.

Request Character Encoding

When an HTTP request is sent to the server, it is encoded in some character encoding by the browser. Unfortunately, modern browsers still do not tell the server explicitly what character encoding the request uses. Therefore, the filter must determine the character encoding of the request. The init-param "encoding" specifies what character encoding to use when processing requests. The default encoding is "UTF-8".

Character encoding is configurable at the tool level. This task is not performed by the portal. Therefore, tools can configure a different character encoding, if desired. This can be done by setting "encoding" to some other character encoding name. Otherwise, the tool can disable the request filter character encoding handling by setting the init-param "encoding.enabled" to "false" on the request filter.

Tomcat Character Encoding

Sakai systems using Tomcat as the Servlet Container need one more configuration to get the character encoding just right. Tomcat interprets the characters in the request path. To control this, Tomcat needs to have a setting in the conf/server.xml, in the connector block which defines the primary http connector used by Sakai. Add the attribute:

```
    URIEncoding="UTF-8"
```

Here's an example:

```
<!-- Define a non-SSL HTTP/1.1 Connector on port 8080 -->
    <Connector port="8080" maxHttpHeaderSize="8192"
               maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
               enableLookups="false" redirectPort="8443" acceptCount="100"
               connectionTimeout="20000" disableUploadTimeout="true"
               URIEncoding="UTF-8" />
```

This is a global setting that effects all request through this connector.

File Upload

The Servlet specification does not specify how file uploads (from a user's browser) are handled on the server side. In Sakai, the request filter handles file uploads using Apache `commons-fileupload` by default. The filter makes the uploaded files available through the request attributes.

Example upload form:

```
<form name="myToolForm" enc="multipart/form-data">
      <p>Upload a file here</p>
      <input type="file" name="uploadedFileFormField" />
</form>
```

Accessing the uploaded file from the example form (from inside a tool):

```
import org.apache.commons.fileupload.FileItem;

...

FileItem item;
item = (FileItem) request.getAttribute("uploadedFileFormField");
String filename = item.getName();
//byte[] contents = item.get();
InputStream contentsAsStream = item.getInputStream();
```

See the Apache commons-fileupload API for details
(http://jakarta.apache.org/commons/fileupload)

A tool can change its maximum allowed upload size using the init-param "upload.max" (in bytes). It can change the threshold and temporary upload directory through the "upload.threshold" and "upload.dir" init-param.

Tools should use the request filter to process file uploads, if possible. However, if a tool must use its own file upload filter instead of Sakai's, it can do so. File upload handling can be disabled by setting the init-param "upload.enabled" to "false". This allows other filters, such as the MyFaces file upload filter, to be used inside of Sakai.

Request Filter Implementation

The request filter is in the `kernel` module's `request` project. It is deployed to shared/lib.

268