Torry Harris

Torry Harris Business Solutions

Extending Struts

Author - Sashwat Gupta

www.thbs.com

Struts is a very powerful and extensible framework. The article discusses two ways of extending struts by creating custom Plug-ins and sub classing the ***RequestProcessor*** class.

Creating custom Plug-Ins

To create a PlugIn, the custom PlugIn class needs to extend the class ***org.apache.struts.action.PlugIn*** and implement two methods, ***init ()*** and ***destroy ()*** which are called at application startup and shutdown, respectively.

A common use of a Plugin Action is to configure or load application specific data as the web application is starting up. It can be used to initialize or cache database connections or objects. Caching resources improves the performance of applications. At runtime, Actions or business tier classes would access any resource setup by init (). The PlugIn interface allows us to setup resources, but does not provide any special way to access them. Most often, the resource would be stored in application context, under a known key, where other components can find it.

```
public class AqPlugin implements PlugIn {

    // the values are set from the struts-config.xml file
    // also include the setters for the fields
    private String mappingFile;
    private String dataFile;

    // This method will be called at application startup time
    public void init(ActionServlet actionServlet, ModuleConfig config)
            throws ServletException {

        try
        {
            ServletContext servletContext = actionServlet.getServletContext();

            // check the database connections/ check for necessary config files etc..
            //here an xml file is being parsed and put in the servlet context
            Data data = (Data) Utils.unmarshall(mappingFile, dataFile);

            // save in application context
            servletContext.setAttribute("PAGE_DETALLE", data);
        }catch (Exception e) {

            throw new ServletException(e);
        }
    }

    // This method will be called at application shutdown time
    public void destroy() {

        // close the connection or do other cleanup operations
    }
}
```

Plug-ins are configured using <plug-in> elements within the Struts configuration file. To configure the plug-in, the configuration details must be added to struts-config.xml file as shown. The set property in the plugin configuration can be used to externalize strings. In the example it has been used to specify the file names.

```
<plug-in className="com.thbs.AqPlugin">
   <set-property property="mappingFile"
                 value="recursos/init/mapping.xml"/>
   <set-property property="dataFile"
                 value="recursos/init/data.xml"  />
</Plug-in>
```

There are two issues to be aware of before going with this approach:

- Multiple requests may have access to the resources, so they need to be thread safe or immutable.
- A server crash may bring down the servlet engine but leave the JVM up. In this condition the Plug-In's method ***destroy()*** might not be invoked.

Both the Tiles and ***Validator*** frameworks use Plug-Ins for initialization by reading configuration files. Two more things, which can be done in a ***PlugIn*** class, are:

- If the application depends on some configuration files, then their availability can be checked in the Plug-In class to throw a ***ServletException*** in case of unavailability. This will result in ***ActionServlet*** becoming unavailable.
- The Plug-In interface's ***init ()*** method is the last chance to make a change in ***ModuleConfig,*** which is a collection of static configuration information that describes a Struts-based module. Struts will freeze ***ModuleConfig*** once all ***PlugIns*** are processed.

Extending RequestProcessor class

ActionServlet is the only servlet in Struts framework, and is responsible for handling all the requests. Whenever it receives a request, it first tries to find a sub-application for the current request. Once a sub-application is found, it creates a ***RequestProcessor*** object for that sub-application and calls its ***process()*** method by passing it ***HttpServletRequest*** and ***HttpServletResponse*** objects.

The ***RequestProcessor.process()*** is where most of the request processing takes place. The process() method is implemented using the Template Method design pattern, in which there is a separate method for performing each step of request processing, and all of those methods are called in sequence from the ***process()*** method.

To create our custom ***RequestProcessor*** the ***RequestProcessor*** class must be extended and the methods whose functionality needs to be modified must be overridden. One of the methods which can be overridden is ***processPreprocess()***. This tells the request processor whether or not to continue processing the request after ***processPreprocess()*** method has been called.

```
public class AqRequestProcessor extends RequestProcessor {

    protected boolean processPreprocess(HttpServletRequest request,
            HttpServletResponse response) {

        // Check if user is logged in.
        // If so return true to continue processing,
        // otherwise return false to not continue processing.
        return (true);

    }
}
```

If the Tiles ***plugin*** is used, then instead of extending the ***RequestProcessor***, ***TilesRequestProcessor*** should be extended. To use a custom request processor, we have to configure Struts to use it in the Struts configuration file.

```
<controller
    ProcessorClass="com.thbs.AqRequestProcessor">
```

The other ways of extending the struts functionality is by subclassing ***ActionServlet*** or ***ActionMapping*** and creating our own taglib by extending the struts taglib classes.