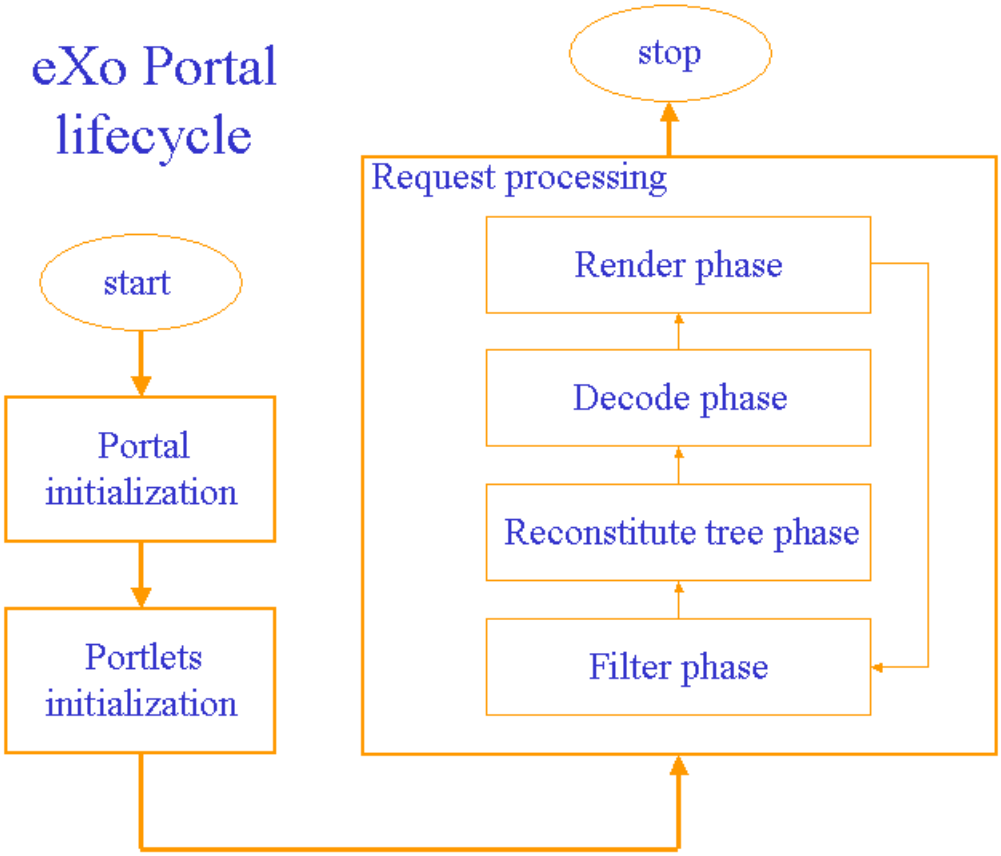
DocV1.PortalRequestLifecycle

Portal request lifecycle

Let's describe how our Java Server Faces based portal works. This section gives an overview of the eXo portal achitecture, how we use JSF and the way the portal interacts with the portlet container. We focus on the request lifecycle which goes through the filter phase, the reconstitute request tree phase, the decode phase and finally, the render phase.

The eXo portal is composed of many portlet WARs and a master eXo portal WAR (several in case of multi portal instances). The eXo portal module is responsible for checking the security, decoding the request, loading the user configuration, building the jsf tree, and distpatching the request to the portlet container. When all the portlets located in the requested page are rendered, the portal returns an aggregated page to the client.

The lifecycle of the exoportal can be seen as :



Note that the eXo portal does not use the entire JSF lifecycle, therefore some phases are not showned in the diagram.

- Portal initialization

When the web server is started or the eXo web arvchive (WAR) is deployed, the PortalContextListener (You can find the configuration of this listener in web.xml) will catch the start event and run the checking code for eXo tables, the default groups and the default users. If the anonymous and admin users are missing, the listener will look up the organization service, to create the missing user.

- Portlet Initialization

When the webserver is started or a portlet war is dropped into the portlet deployed directory, the PortletApplicationListener (You can find the configuration of this listener in portlet.war/WEB-INF/web.xml or default-web.xml) will catch the start event, it will lookup for the portlet container service and register the application with the portlet container if a portlet.xml file is present.

- Request processing

After the server is started and that all the initialization has been done. The portal

is ready to receive the first request. There is a very simple rule to handle the request : any dynamic request must go though the portal and be treated by the filter , the jsf servlet controller and the portlet container. For a static resource request, such as an image file, the request can be handled by the default servlet of portal or the default servlet of the portlet; only depending on the context path of the request.

- Filter phase. When a request is sent to the portal, it will be first handle by either a PublicRequestFilter or a PrivateRequestFilter (you can find the configuration of those 2 filters in exo/WEB-INF/web.xml) depending on the url type. We currently define 2 types of url, public and private. The public url has the form /portal/faces/public… and the private url has the form /portal/faces/private… Both public and private paths you saw in 2 url are virtual paths and they are defined in the web.xml. The role of filter is to check the Owner Context. If the owner context, portal:ctx=user, does not match with the current user context or no user context exists in the session, the filter will destroy the jsf tree in the session, reload the user context according the request, and store the user context in the session.
- Reconstitute JSF tree phase. After the filter phase, the request is forwarded to the FacesServlet. The FacesServlet will get the Lifecycle instance and will execute the faces lifecycle. One important phase in the faces lifecycle is the process reconstitute part. It will check for a tree id in the session - id associated with the request tree - and will reconstruct the component tree if the two trees are not the same. With exo, you always make the request to the same URL so the jsf tree is always cached in session. The JSF tree is destroyed and reconstructed only when the request user context does not match with the one in the session (done in the filter phase). Note that the exo jsf tree is constructed based on the owner-config.xml, owner-navigation.xml and owner-pages.xml configuration file of the user, basically you will have a ui component tree that reflects those xml files.
- JSF Decode phase. The next phase of the faces cycle is the decode phase or apply request phase. In this phase, the JSF implementation iterates over the components in the component tree and calls each component's decode() method. That method extracts information from the request and stores it in the component.

The UIPorlet decode() will do 3 main tasks :

- Check for the "change mode"Â€? event : if the event is detected and the request component id matches the current UIPorlet component id, it will raise an event and will delegate it to the PortletActionListnenter class. The listener will reset the mode in the UIPortlet Component.
- Check for the "change window"Â€? state event: if the event is detected and that the request component id matches the current UIPorlet component id, it will raise an event and will delegate it to

the PortletActionListnenter class. The listener will reset the window state in the UIPortlet Component.
- Check for the portlet action type: According the portlet spec, we have 2 types of action : one is the action type and the other one is the render type. If the type is action , the processAction(..) method of the portlet will be called and then the render(....) method is called. If the type is render, only the render(..) method is called. It is mandatory that the processAction(..) has to be called before any render(..) method is called. The reason for this requirement is because a portlet can process an action and send a message to another portlet. Indeed, it would not make any sense if the render(...) method of the other portlet has already been called.

Once again we can see how JSF technology fits very well with portlet technology, by defining many process phase. This way, it will make sure that each processAction(..) of each portlet will be called first and each render(..) method of each portlet will be called in the render phase

- JSF Render phase. Finally the Render phase creates the html page by calling the methods encodeBegin(..) encodeChildren(..) and encodeEnd(..) of the root component. The parent UIComponent will control the render phase of its children. With the eXo portal jsf tree, the root component is UIPortal component.

In UI Portlet Renderer, since the UIPortlet has no children, only encodeBegin(..) is required to be called. We create the RenderInput object, it contains all the information of the request, and delegates it to the portlet container. The portlet container will then invoke the method render(..) and return an OutputObject. Then the portal renders the portlet header and the portlet body using the content returned by the portlet container in the Output object. Note that in the portal page you have many portlets but each request only targets one portlet. Therefore, the parameter map sent to the container is cached as a parameter map in the associated UIPortlet object. Only the portlet you send request to use HttpServletRequest parameter map or a parameter map produced by the processAction() method. All of those steps are processed in the decode phase.

DocV1.PortalRequestLifecycle (en)
CrÃ©ateur: XWiki.benjamin.mestrallet Creation Date: 2005/02/09 16:22
Dernier Auteur: XWiki.benjamin.mestrallet Last Modification Date: 2005/02/15 22:55