AvalonFortressFAQ ExtendingTheFortressConfiguration

Fortress comes with examples showing the initialization of a Fortress application done by initializing the root container of the application with an instance of class FortressConfig. The elements of the FortressConfig are fixed and each example currently assumes, that they are enough for starting the component and service logic. This is normally true for standalone applications, but if the root container is embedded into an environment like a servlet container or into a plug-in, this might not be enough.

The FortressConfig class creates internally two Context objects, one for the configuration of the root container and one for the default (system) configuration, that is set as the parent context of the first one. Contexts are hierarchical. If a key is not found in the current context, it is requested from its parent. Therefore we have to create our own Context object, if we want to support additional configuration parameters from the embedding application.

FortressConfig has an additional constructor to define the parent context on our own. It is used to set our own context as parent of the Fortress root container's context. The easiest solution is to create an instance of DefaultContext and initialize it with a populated map. Do not forget to set the parent of your own context. This should still be the default configuration created with FortressConfig.createDefaultContext.

See example code used in your servlets init method:

```
   public void init(final ServletConfig pServletConfig) throws ServletException
    {
        ...
        final DefaultContext lContext =
            new DefaultContext(initializeMap(),FortressConfig.createDefaultConfig());
        lContext.makeReadOnly();
        final FortressConfig lConfig = new FortressConfig(lContext);
        ...
    }
    private Map initializeMap()
    {
        Map lMap = new Hashtable();
        lMap.put(CACHE_KEY, getServletConfig.getInitParameter("enable-cache"));
        return lMap;
    }
    private static String CACHE_KEY = "app.cache";
```

You might add also additional services provided by the servlet context istelf i.e. a servive for mime types. Assuming you have an interface for this:

```
public interface MimeTypeService
{
    public static final String ROLE = MimeTypeService.class.getName();
    /** Request a mime type for a file name. */
    public String get(final String pFilename);
}
```

and an implementation for a servlet environment:

```
public class ServletMimeTypeService implements MimeTypeService
{
    private final ServletContext m_ServletContext;
    public ServletMimeTypeService(final ServletContext pServletContext)
    {
        m_ServletContext = pServletContext;
    }
    public String get(final String pFilename)
    {
        return m_ServletContext.getMimeType(pFilename);
    }
}
```

you can make this service available for your Fortress container by using an own DefaultServiceManager and put this into the FortressConfig during the servlet init:

```
public void init(final ServletConfig pServletConfig) throws ServletException
{
    ...
    final ServletContext lServletContext = pServletConfig.getServletContext();
    final DefaultServiceManager lServiceManager = new DefaultServiceManager();
    lServiceManager.put(MimeTypeService.ROLE, new ServletMimeTypeService(lServletContext);
    final FortressConfig lConfig = new FortressConfig();
    lConfig.setServiceManager(lServiceManager);
    ...
}
```

Naturally you can combine this with your own context described above.