Servlet Embedded NetKernel

This is the Beta 1 release of the Servlet embedded version of NetKernel.

The Servlet embedded version of NetKernel creates and boots an instance of NetKernel within a JEE Servlet container allowing Servlet based applications to utilize the flexibility, caching and other features of NetKernel within and alongside new and existing more traditional applications.

There are three files in this distribution, netkernelservlet.jar, web.jar and timeservice.war.

The timeservice.war file is a complete ready-to-use web application which implements a RESTful time web service. To use this, place the timeservice.war file in the directory of your container that accepts deployable WAR files. For example, with Jetty this is jetty/webapps/. After deploying this web application, use your browser to request http://localhost:8080/timeservice/ (or a different port if that is how your container is configured) and the home page for the service will be displayed. That home page contains instructions on how to use the service.

The netkernelservlet.jar file contains the NetKernel Servlet. This is a traditional servlet that can be installed in the WEB-INF/lib/ directory of a web application and configured in the web.xml file as shown next.

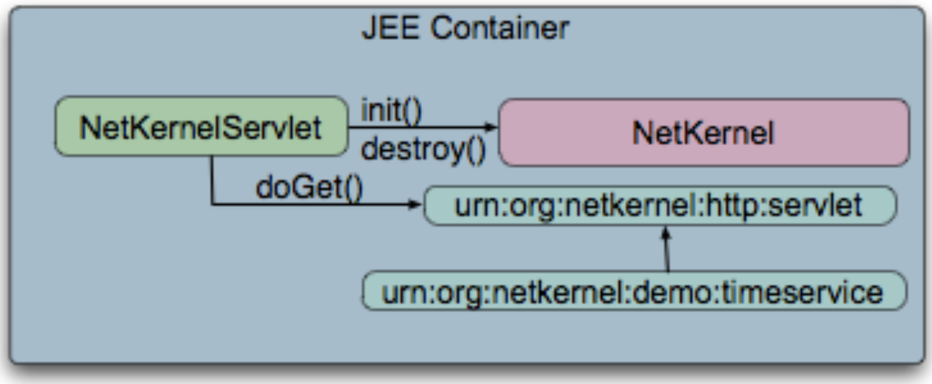
```
  <servlet>
    <description>Front end for embedded NetKernel</description>
    <display-name>NetKernel Servlet</display-name>
    <servlet-name>NetKernelServlet</servlet-name>
    <servlet-class>org.ten60.netkernel.servlet.NetKernelServlet</servlet-class>
    <load-on-startup>0</load-on-startup>
  </servlet>
```

It is recommended that the servlet be configured using the <load-on-startup> element to ensure it is created when the container starts. A value of zero (0) will ensure it loads before other services.

The URL mapping can be set so that all requests are sent to the servlet, as shown in the following web.xml fragment:

```
 <servlet-mapping>
    <servlet-name>NetKernelServlet</servlet-name>
    <url-pattern>/*</url-pattern>
  </servlet-mapping>
```

The role of NetKernelServlet is to both manage an instance of embedded NetKernel and to create root requests triggered from incoming Servlet requests. The following diagram illustrates the relationships.


JEE Container
NetKernelServlet
init()
destroy()
NetKernel
doGet()
urn:org:netkernel:http:servlet
urn:org:netkernel:demo:timeservice

When the servlet is loaded by the container its init() method is called and the following occurs:

- An instance of NetKernel is created
- Any kernel parameters set as init-param elements are read and used
- The file modules.conf is processed; each listed module is added to the module manager
- NetKernel is set to run level 1, which causes all modules to be commissioned
- A transport is created that will issue requests to the module urn:org:netkernel:servlet:http.

When any of the various do...() methods (e.g. doGet()) are called on NetKernelServlet, the servlet creates a root request which is issued to the module urn:org:netkernel:servlet:http, which contains the ServletBridge overlay.

When the servlet's destroy() method is called, the servlet shuts down the instance of NetKernel.

In order for NetKernelServlet to boot NetKernel, the following JAR file must be placed in the web application WEB-INF/lib/ directory in order for the JEE classloader to find and load them:

```
urn.com.ten60.core.cache.se-1.x.x.jar
urn.com.ten60.core.layer0-1.x.x.jar
urn.com.ten60.core.module-standard-1.x.x.jar
urn.com.ten60.core.netkernel.api-4.x.x.jar
urn.com.ten60.core.netkernel.impl-4.x.x.jar
```

Any other modules required by NetKernel or your application must be loaded by the NetKernel classloaders and so *must not* be placed in the WEB-INF/lib/ directory. Instead, these modules are place in a modules/ directory at the root of your web application. In addition the file modules.conf must be located at the root of your web application and will contain a list of these modules that are to be loaded and commissioned by NetKernel when it boots. The following is an example of a modules.conf file (note that the correct version of each JAR file will replace the "x.x.x" part of the file names).

```
# list of module directory or JAR file names that are to be loaded for core NetKernel
urn.org.netkernel.ext.layer1-1.x.x.jar
urn.org.netkernel.ext.system-1.x.x.jar
urn.org.netkernel.xml.core-1.x.x.jar
urn.org.netkernel.lang.groovy-1.x.x.jar

# Module for NetKernel Servlet adapter
urn.org.netkernel.servlet.http/

# Application modules next
urn.org.netkernel.demo.timeservice/

# These modules are required for the BEF support
urn.org.netkernel.tpt.http-1.x.x.jar
urn.org.netkernel.fulcrum.backend-1.x.x.jar
urn.org.netkernel.ext.system-1.x.x.jar
urn.org.netkernel.nkse.style-1.x.x.jar
urn.org.netkernel.nkse.control.panel-2.x.x.jar
urn.org.netkernel.ext.introspect-1.x.x.jar
urn.org.netkernel.lang.xrl-1.x.x.jar
urn.org.netkernel.mod.visualizer-1.x.x.jar
```

Note that the modules are grouped into four sections. The first section lists the modules that are required for NetKernel to boot and run. The second section contains the ServletFulcrm module. The third section contains your application modules (in this case we show the Time Service demo module). The fourth section contains the modules required to run the BackendFulcrum. If these modules are included, NetKernel will boot up the BEF and open up TCP/IP port 1060. You can use most of the tools as you would in the NetKernel Standard Edition distribution. Of particular value is the Visualizer. (For a production release you would omit the fourth section).

The Servlet Fulcrum can be used just like the FrontendFulcrum. You can have your modules imported into this fulcrum by providing the resource /etc/system/SimpleDynamicImportHook.xml with the following contents:

```
<connection>
  <type>ServletFulcrum</type>
</connection>
```

From the perspective of you NetKernel modules, they will not know they are running in the embedded version. Requests from the Servlet will be issued to res:/... For example, the Time Service root URL is http://localhost:8080/timeservice. That is issued into the implementation module as res:/timeservice/ .