GBeans

Stale Content

The content of this page is growing stale and may or may not contain relevant, useful or correct information.

Overview

A GBeanMBean is a J2EE Management Managed Object, and is standard base for Geronimo services. GBean is a class or object that Geronimo the container can manage the lifecycle of. So, for example, when you deploy parts into a container you want to start them and stop them, and they have dependencies: Such as 'Bean A' can only start up when 'Bean B' is ready to go because 'Bean B' provides services that 'Bean A' needs. So, GBeans are Geronimo's way of packaging up things are that need to be managed, and can express dependencies. The GBeans architecture lies at the heart of Geronimo's power to enable developers to move or work with their existing J2EE assets, whether Open Source or commercial.

GBeans are designed to let you take things you have, put a GBean wrapper around them, and use that to bridge the JSR 77 lifecycle requirements which GBeans support). You can take anything lying around and get it to work with GBeans. In addition, the GBeans will let developers bring other existing Open Source Java projects into the Geronimo orbit.

This wraps one or more target POJOs and exposes the attributes and operations according to a supplied GBeanInfo instance. The GBeanMBean also supports caching of attribute values and invocation results which can reduce the number of calls to a target.

- Gbeans are way of providing managements services to Geronimo
- It is implemented on top of the JMX right now but supposes to be independent of JMX. User should be able to use them without knowledge of the JMX. And it can be make independent of GBeans .
- The GBean can be converted into the MBean using the GBeanMBean and use the JMX to give the services

Sample GBeans

Grab the source code and search for the implementators of the GBeanLifecycle. It is a good way to find GBeans examples. It's been found that the jetty module and the GBeanTest class are quite helpful.

TODO: Add the fully qualified class names

How to write GBean

- A GBean may implement the optional org.apache.geronimo.gbean.GBeanLifecycle interface. When a GBean implements this interface, the implementation will get lifecycle callbacks.
- They can have attributes with getters and setters. There are attributes that are final and I think they are standard .. (What is the list of them) e.g. name, Kernel (they can use to get hold of the system info, have a look at the sample GBeans.) There are ordinary attributes .. have getters and setters
- GBeans can have methods ..
- A GBean's constructor ***MUST NOT*** call its own member functions which are declared in its GBeanInfo!!!
- A GBean MUST implement a method having the following signature:

```
public static GBeanInfo getGBeanInfo()
```

This method provides meta-data about the attributes, operations and references exposed or used by the GBean. It is standard practice (a Geronimo pattern) to initialize this meta-data in a static block initializer. The following block depicts such an approach:

-
```
public static final GBeanInfo GBEAN_INFO;

static {
    GBeanInfoBuilder infoBuilder = new GBeanInfoBuilder("AxisGbean", AxisGbean.class);
    infoBuilder.addAttribute("name", String.class, true);
    infoBuilder.addAttribute("kernel", Kernel.class, false);
    infoBuilder.addOperation("echo", new Class[]{String.class});
    infoBuilder.setConstructor(new String[] {"kernel","Name"});
    GBEAN_INFO = infoBuilder.getBeanInfo();
}
```

Attributes and references MUST be compliant wit the following naming conventions: attributes MUST start with a lower case first character; and references must should start with an upper case first character. This simple naming convention should simplify the configuration of a GBean.

Step 1: Your first simple GBean

```
package example1;

import org.apache.geronimo.gbean.GBeanInfo;
import org.apache.geronimo.gbean.GBeanInfoBuilder;

public class MyGBean {

    public static final GBeanInfo GBEAN_INFO;

    static {
        GBeanInfoBuilder infoBuilder = new GBeanInfoBuilder("MyGBean", MyGBean.class);

        GBEAN_INFO = infoBuilder.getBeanInfo();
    }

    public static GBeanInfo getGBeanInfo() {
        return GBEAN_INFO;
    }
}
```

Compile the class using the following command (**TODO**: create a maven task to compile a user's gbeans; simply find a better way).

```
$ javac -classpath <geronimo_home>/assembly/target/geronimo-1.0-SNAPSHOT/lib/geronimo-kernel-1.0-SNAPSHOT.jar example1/MyGBean.java
```

TODO: Change what follows in this section (temporarily useful)

Create a jar with the gbean class and place it into <geronimo_home>/assembly/target/geronimo-1.0-SNAPSHOT/repository/example1.

```
$ jar -Mcvf mygbean.jar example1/*.class
$ mkdir <geronimo_home>/assembly/target/repository
$ cp mygbean.jar <geronimo_home>/assembly/target/repository
```

How to deploy GBean

Deploying a GBean onto Geronimo requires to create a specialised Geronimo plan (aka configuration). It's similar to J2EE deployment descriptors as it also describes what it looks like and what references to other GBean it must have started before it starts up.

The plan has to conform to *schema/geronimo-config.xsd* (in the repo it's in *modules/deployment/src/schema/geronimo-config.xsd*)

Here is a part of the deployment descriptor for the gbeans already available in the binary distribution of Geronimo (it's *modules/assembly/src/plan/j2ee-server-plan.xml* in the repo). It looks really scary if this is your first time seeing one. Down below we have a simple one that we'll use for your GBean.

```
    <gbean name="openejb:type=ContainerIndex" class="org.openejb.ContainerIndex">
        <references name="EJBContainers">
            <pattern>geronimo.server:j2eeType=StatelessSessionBean,*</pattern>
            <pattern>geronimo.server:j2eeType=StatefulSessionBean,*</pattern>
            <pattern>geronimo.server:j2eeType=EntityBean,*</pattern>
        </references>
    </gbean>

    <!-- EJB Protocol -->
    <gbean name="openejb:type=SocketService,name=EJB" class="org.openejb.server.SimpleSocketService">
        <attribute name="serviceClassName" type="java.lang.String">org.openejb.server.ejbd.EjbServer<
/attribute>
        <attribute name="onlyFrom" type="java.net.InetAddress[]">127.0.0.1</attribute>
        <reference name="ContainerIndex">openejb:type=ContainerIndex</reference>
    </gbean>
    <gbean name="openejb:type=ServiceDaemon,name=EJB" class="org.openejb.server.ServiceDaemon">
        <attribute name="port" type="int">4201</attribute>
        <attribute name="inetAddress" type="java.net.InetAddress">127.0.0.1</attribute>
        <reference name="SocketService">openejb:type=SocketService,name=EJB</reference>
    </gbean>

    <!-- JSR77 Management Objects -->
    <gbean name="geronimo.server:j2eeType=J2EEDomain,name=geronimo.server" class="org.apache.geronimo.
j2ee.management.impl.J2EEDomainImpl"/>
    <gbean name="geronimo.server:j2eeType=J2EEServer,name=geronimo" class="org.apache.geronimo.j2ee.
management.impl.J2EEServerImpl">
        <reference name="ServerInfo">geronimo.system:role=ServerInfo</reference>
    </gbean>
    <gbean name="geronimo.server:j2eeType=JVM,J2EEServer=geronimo" class="org.apache.geronimo.j2ee.
management.impl.JVMImpl"/>

    <!-- JMX Remoting -->
    <gbean name="geronimo.server:role=JMXService,name=localhost" class="org.apache.geronimo.jmxremoting.
JMXConnector">
        <attribute name="URL" type="java.lang.String">service:jmx:rmi://localhost/jndi/rmi:/JMXConnector<
/attribute>
        <attribute name="applicationConfigName" type="java.lang.String">JMX</attribute>
    </gbean>
```

See also **BROKEN INTERWIKI LINK to wiki:Deployment#head-5cbd584046863bc7b753e57e8681a98a87f36f0f (label = Service configuration and deployment)**.

TODO: Describe what these elements mean (here or even better in xsd)

So that was scary. The next section describes the deployment plan necessary for your GBean. It's really not so bad...

Step 2: Your first deployment plan

Here's the plan of your first simple GBean - MyGBean. The gbean doesn't expose any attributes or have references to other GBean.

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration
    xmlns="http://geronimo.apache.org/xml/ns/deployment"
    configId="example1/MyGBean"
    >

    <dependency>
        <uri>mygbean.jar</uri>
    </dependency>

    <gbean name="geronimo.example:name=My first simple GBean" class="example1.MyGBean" />

</configuration>
```

There are two ways to deploy the GBean to your server, the so-called "offline" mode for when your server isn't running, and the "online" mode, when it is. Both are outlined below. Deploying the gbean requires to execute *Geronimo Deployer*. It boils down to executing an executable jar *bin/deployer.jar*.

Offline Deployment

When the Geronimo server isn't running, it's possible to add a GBean to it's configuration, and then start the GBean later when the server is running.

First, 'distribute' the GBean to the server :

- ```
  java -jar bin/deployer.jar distribute mygbean-plan.xml
  ```

This will do all the necessary things to get the GBean to the server, but the GBean won't be running when the server starts.

Now, start the server :

- ```
  java -jar bin/server.jar
  ```

And when that is complete, start your GBean :

- ```
  java -jar bin/deployer.jar start example1/MyGBean
  Username: system
  Password: manager
  ```

You'll be prompted for the username and password in order to start the GBean, as shown above. Use the values shown above.

Note that unless you start the server "in the background", you'll need another command prompt to start your GBean.

After starting, you should see the following in your server log :

- ```
  11:43:11,652 INFO  [ConfigurationManagerImpl] Loaded Configuration geronimo.config:name="example1
  /MyGBean"
  11:43:11,717 INFO  [Configuration] Started configuration example1/MyGBean
  ```

Online Deployment

When you already have a server running, you can distribute and start the GBean in one step :

- ```
  java -jar bin/deployer.jar deploy  mygbean-plan.xml
  Username: system
  Password: manager
  ```

And you'll see a slightly different message in the log.

- ```
  11:44:25,340 INFO  [LocalConfigStore:config-store] Installed configuration example1/MyGBean in location
  19
  11:44:25,432 INFO  [ConfigurationManagerImpl] Loaded Configuration geronimo.config:name="example1
  /MyGBean"
  11:44:25,446 INFO  [Configuration] Started configuration example1/MyGBean
  ```

How to run GBean

Step 3: Your first GBean in action

Once the deployment completes, start the following command from the Geronimo home directory:

- ```
  java -jar bin/server.jar example1/MyGBean
  ```

You should see the following output on the console:
  ```

```
$ java -jar bin/server.jar example1/MyGBean
15:29:42,376 WARN  [ToolsJarHack] Could not all find java compiler: lib\tools.jar file not found in C:
\Program Files\Jav
a\j2re1.4.2_05 or C:\Program Files\Java
15:29:42,386 INFO  [Daemon] Server startup begun
15:29:43,230 INFO  [Kernel] Starting boot
15:29:43,612 INFO  [MBeanServerFactory] Created MBeanServer with ID: 17ce4e7:ff076f1f39:-8000:JLASKOWSKI:
1
15:29:43,833 INFO  [Kernel] Booted
15:29:43,963 INFO  [ConfigurationManagerImpl] Loaded Configuration geronimo.config:name="org/apache
/geronimo/System"
15:29:44,586 INFO  [Configuration] Started configuration org/apache/geronimo/System
15:29:45,048 INFO  [RMIRegistryService] Started RMI Registry on port 1099
15:29:45,139 INFO  [ReadOnlyRepository] Repository root is file:/C:/projects/geronimo/trunk/modules
/assembly/target/geronimo-1.0-SNAPSHOT/repository/
15:29:45,229 INFO  [ConfigurationManagerImpl] Loaded Configuration geronimo.config:name="MyGBean"
15:29:45,249 INFO  [Configuration] Started configuration MyGBean
15:29:45,249 INFO  [Daemon] Server startup completed
```

The line *Loaded Configuration geronimo.config:name="MyGBean"* indicates that your first GBean is really deployed and running! Hurray!

You may also want to see some information about the gbean in the Geronimo Debug Console. Start the server with the following command:

```
java -jar bin/server.jar org/apache/geronimo/DebugConsole example1/MyGBean
```

and open up http://localhost:8080/debug-tool/index.vm?ObjectNameFilter=*%3Aname%3D%22MyGBean%22&MBeanName=geronimo.config%3Aname%3D%22MyGBean%22 in your browser.

And here is how to run the gbean programatically:

Having created a GBean like this to start/stop/set values/invoke operations and GBean service use code like follows

```
name = new ObjectName("test:name=AxisGBean");
kernel = new Kernel("test.kernel", "test");
kernel.boot();
ClassLoader cl = getClass().getClassLoader();
ClassLoader myCl = new URLClassLoader(new URL[0], cl);
GBeanMBean gbean = new GBeanMBean(AxisGbean.getGBeanInfo(), myCl);
gbean.setAttribute("name", "Test");

kernel.loadGBean(name, gbean);
Kernel.startGBean(name);
System.out.println(kernel.getMBeanServer().getAttribute(name, "state"));
System.out.println(kernel.getMBeanServer().invoke(name, "echo", new
Object[]{"Hello"}, new String[]{String.class.getName()}));

kernel.stopGBean(name);
kernel.unloadGBean(name)
```

Other questions to be answered later

If you feel you can answer some of the questions please do so!

What is GBean Meta Data?

What does the GBeanInfo class represent?

Can I configure my GBean using a flat file?

How do I manage the attributes of my GBean remotely?

GBeans are exposed as MBeans via the JMX kernel. Hence, it is possible to control/query them as we would have control Ã,"standardÃ," MBeans.

For instance, to manage a given MBean remotely, it is possible to leverage the Java Management Extensions Remote API(JSR 160), which is supported by Geronimo.

More accurately, two services need to be up and running in order to enable the RMI Connector defined by JSR 160 on the server-side:

- *org.apache.geronimo.system.RMIRegistryService*: starts a RMI registry on the specified port. This service is part of the "system" plan, in other words it is always started; and
- *org.apache.geronimo.jmxremoting.JMXConnector*: it creates a RMI Connector server and exports it to the RMI registry embedded in its URL (see JSR 160 for more details about the format of the URL). You can have a look to the *j2ee-server* plan for more details on its GBean definition.

If the two above services are running, then the following snippet will get you an *MBeanServerConnection*:

```
        Map environment = new HashMap();
        String[] credentials = new String[]{<username>, <password>};
        environment.put(JMXConnector.CREDENTIALS, credentials);

        JMXServiceURL address = new JMXServiceURL(<URI defined by the JMXConnector  service>);
        JMXConnector jmxConnector = JMXConnectorFactory.connect(address, environment);
        MBeanServerConnection mbServerConnection = jmxConnector.getMBeanServerConnection();
```

Note: *username* and *password* must be defined by the *org.apache.geronimo.security.jaas.ConfigurationEntry* having the name defined by the *applicationConfigName* attribute of the *org.apache.geronimo.jmxremoting.JMXConnector* service.

Having said that, if you do not want to write code, you can also use a JMX console supporting JSR 160.

How do I call methods on a GBean deployed in the server from a J2EE App?

First get a handle to the kernel by calling KernelRegistry.getSingleKernel() and then use the invoke methods in the Kernel interface. eg:

```
        ObjectName obj = new ObjectName("geronimo.example:name=EchoServer");
        Kernel kernel = KernelRegistry.getSingleKernel();
        String outputs=(String)kernel.invoke(obj,"hello");
```

where hello() is a method on the GBean. This will work provided that the GBean is running in the same server instance where the J2EE application is deployed.

Article about GBeans

Geronimo GBeans Architecture