Managing Restcomm JAIN SLEE

Table of Contents

JAIN SLEE JMX Agent 1
SNMP Agent 1
Managing JAIN SLEE Components 1
Persistent Deployable Unit Management 1
  Persistent Deployable Unit Install 1
  Persistent Deployable Unit Uninstall 1
  Beyond Deployable Unit (Un)Install 2
  Deploy-Config Extension 2
Ant Tasks 4
  SLEE Management Task 5
Management Consoles 8
  JMX Console 8
  SLEE Management Console 9
  Jopr Console 9
  TWIDDLE CLI 9

JAIN SLEE JMX Agent

The JMX Agent exposes all MBeans running in the server, including the ones mandated by the JAIN SLEE 1.1 specification.

The operations done through the JMX Agent are not kept once the server is shutdown. For instance, if a deployable unit is installed through JMX, and the server is shutdown, once the server is started again the deployable unit will not be installed.

By default, the JMX Agent listens to port 1099, and is only available at the host/ip which the server is bound.

SNMP Agent

JBoss Application Server provides an SNMP Agent, which can be used to ...

TODO

Managing JAIN SLEE Components

Persistent Deployable Unit Management

JAIN SLEE provides a file deployer that greatly simplifies the management of JAIN SLEE deployable unit jars. The deployer:

- Handles the installation of enclosed JAIN SLEE components.
- Automatically activates and deactivates services contained.
- Handles the creation, removal, activation, deactivation, link binding and link unbinding of all Resource Adapter Entities.

All operations are persistent, which means that unlike management done through JMX, these survive server shutdowns.

Persistent Deployable Unit Install

To install a deployable unit jar simply copy the file to *$JBOSS_HOME/server/profile_name/deploy/*, where `profile_name` is the server profile name. Child directories can be used.

Persistent Deployable Unit Uninstall

To uninstall a deployable unit jar simply delete the file.

Beyond Deployable Unit (Un)Install

The file deployer provides additional behavior then simply (un)install deployable unit jars, as done through the JAIN SLEE 1.1 DeploymentMBean:

Service (De)Activation

All services contained in the deployable unit jar are activated after the install, and deactivated prior to uninstall. On service activation, if there is an active service in the SLEE, with same service name and vendor, then it is considered an older version, and the SLEE deactivates it. The deactivation of the old, and activation of the new, is done smoothly in a single operation, allowing service upgrades with no down time.

Dependencies Management

The deployer puts the installation process on hold until all of the component's dependencies are installed and activated. When uninstalling, it waits for all of the components which depend on components inside the deployable unit to be uninstalled. After an install or uninstall, the deployer evaluates all operations on hold.

Deploy-Config Extension

A deployable unit jar may include a deploy-config.xml file in its *META-INF/* directory. This file provides additional management actions for persistent install/uninstall operations:

Resource Adaptor Entity Management

It is possible to specify RA entities, and the container will create and activate the RA entities after the deployable unit is installed. During the uninstall process, the container will deactivate and remove those RA entities.

Resource Adaptor Links Management

It is possible to specify RA links, and the container will bind those after the deployable unit is installed. When uninstalled, the container will unbind those RA links as well. The links are set for resource adapter entities created in the *deploy-config.xml* file.

This file should comply with the following schema:

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

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="deploy-config">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="ra-entity" maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="property">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string" use="required" />
      <xs:attribute name="type" type="xs:string" use="required" />
      <xs:attribute name="value" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="properties">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="property" maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="file" type="xs:string" use="optional" />
    </xs:complexType>
  </xs:element>

  <xs:element name="ra-entity">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="properties" maxOccurs="1" minOccurs="0"/>
        <xs:element ref="ra-link" maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="resource-adaptor-id" type="xs:string" use="required" />
      <xs:attribute name="entity-name" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>

  <xs:element name="ra-link">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string" use="required" />
    </xs:complexType>
  </xs:element>

</xs:schema>
```

```
<ra-entity
    resource-adaptor-
id="ResourceAdaptorID[name=JainSipResourceAdaptor,vendor=net.java.slee.sip,version=1.2
]"
    entity-name="SipRA">
    <properties>
        <property name="javax.sip.PORT" type="java.lang.Integer" value="5060" />
    </properties>
    <ra-link name="SipRA" />
</ra-entity>
```

The deploy-config.xml example above defines a resource adaptor entity named `SipRa`, to be created for the resource adaptor with id `ResourceAdaptorID[name=JainSipResourceAdaptor, vendor=net.java.slee.sip, version=1.2]`, and with a single config property named `javax.sip.PORT` of type `java.lang.Integer` and with value `5060`. Additionally, a resource adaptor link named `SipRa` should be bound to the resource adaptor entity.

After the deployable unit is installed, the resource adaptor entity is created, activated and the resource adaptor link is bound. Before the deployable unit is uninstalled, or the server is shutdown, the link is unbound, then the resource adaptor entity is deactivated, and finally the same resource adaptor entity is removed.

The SLEE includes a deploy-config.xml file at *$JBOSS_HOME/server/profile_name/deploy/restcomm-slee*, where `profile_name` is the server profile name, and that file can be used to specify RA entities and links too. If an RA is installed using the persistent deployer, then SLEE reads its own deploy-config.xml file, and if there are RA entities and/or links specified with same RA ID, then the operations to create/activate/deactivate/remove these are executed too, as if these were specified in the Deployable Unit's own deploy-config.xml.

Ant Tasks

JAIN SLEE includes some tasks for Apache Ant, which can be used for common management tasks done through , when the container is running. The tasks come bundled in *$JBOSS_HOME/server/default/deploy/restcomm-slee/lib/ant-tasks.jar*. To use these, the Ant script must include the following code:

```
<property environment="system" />
<property name="node" value="default" />
<property name="jboss.deploy" value="${system.JBOSS_HOME}/server/${node}/deploy" />

<path id="project.classpath">
    <fileset dir="${jboss.deploy}/restcomm-slee/lib">
        <include name="*.jar" />
    </fileset>
    <fileset dir="${system.JBOSS_HOME}/client">
        <include name="*.jar" />
    </fileset>
</path>

<property name="project.classpath" refid="project.classpath" />

<property name="jnpHost" value="127.0.0.1" />
<property name="jnpPort" value="1099" />

<taskdef name="slee-management"
    classname="org.mobicents.ant.MobicentsManagementAntTask"
    classpath="${project.classpath}" />
```

It is important to understand some properties set by the code above:

`node`
: This property defines the server configuration profile to be used, for further information about those refer to [_server_profiles].

`jnpHost`
: The host/ip which Restcomm JAIN SLEE is bound.

`jnpPort`
: The port which the JMX Agent is listening.

The property values can be overridden when invoking the Ant script. To do this, use the parameter `-DpropertyName=propertyValue`. For example, the server profile can be changed from `default` to `all` using `-Dnode=all`.

SLEE Management Task

The `slee-management` task allows a script to manage JAIN SLEE deployable units, services and resource adapters. The operations, or sub-tasks, are done through .

```
<slee-management jnpport="${jnpPort}" host="${jnpHost}">
    <!-- sub-tasks -->
</slee-management>
```

The attributes have the same meaning as the properties listed in the script code to import the tasks here: Ant Tasks.

Install Deployable Unit Sub-Task

The `install` sub-task installs JAIN SLEE the deployable unit jar pointed by the value of attribute `url`. Example of usage:

```
<slee-management jnpport="${jnpPort}" host="${jnpHost}">
    <install url="file:///tmp/my-deployable-unit.jar" />
</slee-management>
```

Uninstall Deployable Unit Sub-Task

The `uninstall` sub-task uninstalls JAIN SLEE the deployable unit jar which was installed from the value of attribute `url`. Example of usage:

```
<slee-management jnpport="${jnpPort}" host="${jnpHost}">
    <uninstall url="file:///tmp/my-deployable-unit.jar" />
</slee-management>
```

Activate Service Sub-Task

The `activateservice` sub-task activates an already installed JAIN SLEE service, with the id specified by the value of attribute `componentid`. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
    <activateservice
        componentid="ServiceID[name=FooService,vendor=org.mobicents,version=1.0]" />
</slee-management>
```

Deactivate Service Sub-Task

The `deactivateservice` sub-task deactivates an already installed JAIN SLEE service, with the id specified by the value of attribute `componentid`. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
    <deactivateservice
        componentid="ServiceID[name=FooService,vendor=org.mobicents,version=1.0]" />
</slee-management>
```

Create Resource Adaptor Entity Sub-Task

The `createraentity` sub-task creates the resource adaptor entity with the name specified by the value of attribute `entityname`, for an already installed JAIN SLEE resource adaptor, with the id

specified by the value of attribute `componentid`. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
    <createraentity
        componentid="ResourceAdaptorID[name=FooRA,vendor=org.mobicents,version=1.0]"
        entityname="FooRA" />
</slee-management>
```

Remove Resource Adaptor Entity Sub-Task

The `removeraentity` sub-task removes the resource adaptor entity with the name specified by the value of attribute `entityname`. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
    <removeraentity entityname="FooRA" />
</slee-management>
```

Activate Resource Adaptor Entity Sub-Task

The `activateraentity` sub-task activates the resource adaptor entity with the name specified by the value of attribute `entityname`. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
    <activateraentity entityname="FooRA" />
</slee-management>
```

Deactivate Resource Adaptor Entity Sub-Task

The `deactivateraentity` sub-task deactivates the resource adaptor entity with the name specified by the value of attribute `entityname`. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
    <deactivateraentity entityname="FooRA" />
</slee-management>
```

Bind Resource Adaptor Link Sub-Task

The `bindralinkname` sub-task binds the resource adaptor link with the name specified by the value of attribute `linkname`, for an already active JAIN SLEE resource adaptor entity, with the name specified by the value of attribute `entityname`. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
    <bindralinkname entityname="FooRA"
        linkname="FooRA" />
</slee-management>
```

Unbind Resource Adaptor Link Sub-Task

The `unbindralinkname` sub-task unbinds the resource adaptor link with the name specified by the value of attribute `linkname`. Example of usage:

```
<slee-management host="${jnpHost}" jnpport="${jnpPort}">
    <unbindralinkname linkname="FooRA" />
</slee-management>
```

Management Consoles

JMX Console

JBoss Application Server provides a simple web console that gives quick access to all MBeans registered in the server, which includes the ones defined by the JAIN SLEE 1.1 specification.

To access the JMX console once the server is running, point a web browser to `http://ip:8080/jmx-console`, where `ip` is the IP/Host the container is bound. Unless set during start up, the IP/Host will be `127.0.0.1/localhost` by default.

MBeans in the domain `javax.slee` are all standard JAIN SLEE 1.1 MBeans, while the ones in the domain `org.mobicents.slee` are proprietary to Restcomm JAIN SLEE. The following ones are of particular interest:

`org.mobicents.slee:service=MobicentsManagement`
: the MBean which can be used to make non persistent changes to the server configuration, in runtime. The operation `dumpContainerState` displays a textual snapshot of the server's state, which can be used to quickly look for memory leaks or other debug/profiling related tasks.

`org.mobicents.slee:name=DeployerMBean`
: the MBean allows interaction with the persistent deployable unit deployer. The operation `showStatus` displays a textual snapshot of the deployers's state, which can be used to quickly find out if there is any deployable unit deployment pending, for instance, due to missing dependencies.

`org.mobicents.slee:name=CongestionControlConfiguration`
: the MBean allows changing or retrieving the Congestion Control feature, with the container running. Details are provided in section [_congestion_control_configuration].

For further information about JAIN SLEE 1.1 MBeans and their operations refer to the JAIN SLEE 1.1 Specification, all are covered with great detail.

SLEE Management Console

The JMX Console is simple but the MBeans operations were made considering its invocation by management clients, not people using browsers. The SLEE Management Console is a web application that provides high level management functionality for the SLEE, and comes pre-deployed in SLEE binary releases. To access this console point a web browser to `http://ip:8080/slee-management-console`, where `ip` is the IP/Host the container is bound. Unless set during start up, the IP/Host will be `127.0.0.1/localhost` by default.

Full documentation for this management tool can be found in *docs/tools/slee-management-console* directory.

Jopr Console

Joprwas developed to become Red Hat's Middleware Administration Tool, providing an unified interface and extensible model, to be used mainly to control and monitor servers individually, or clusters.

Restcomm JAIN SLEE binary release includes a JoprConsole in *tools/jopr-plugin*, with standalone documentation on same path, but inside *docs* directory.

TWIDDLE CLI

Both, Console and Jopr Console, are graphic(web) based tools. Some deployments may require command line access to Restcomm . To aid such cases, Restcomm offers `TWIDDLE` based CLI. It allows to manage single instance (remote or local) of Restcomm server.

Restcomm JAIN SLEE binary release includes a TWIDDLE CLI in *tools/twiddle*, with standalone documentation on same path, but inside *docs* directory.