Advanced Topics

Table of Contents

Class Loading

Each JAIN SLEE Component has its own classloader (`ComponentClassLoader`) in the package named `org.mobicents.slee.container.component.deployment.classloading`. This classloader sees the component classes contained in the component jar (`URLClassLoaderDomain`) by declaring it as the parent classloader, and adding the classes seen by each component it refers. It does not see classes from a component that it does not depend on.

JAIN SLEE defines a class loading domain with the s required in the JAIN SLEE 1.1 container (for example, JAIN SLEE, and). This domain (`JBoss Microcontainer ClassLoadingDomain`) imports all classes shared in the JBoss Application Server , and acts like the parent domain for all `URLClassLoaderDomain`s, which means that a class imported by a SLEE classloading domain will always be used first.

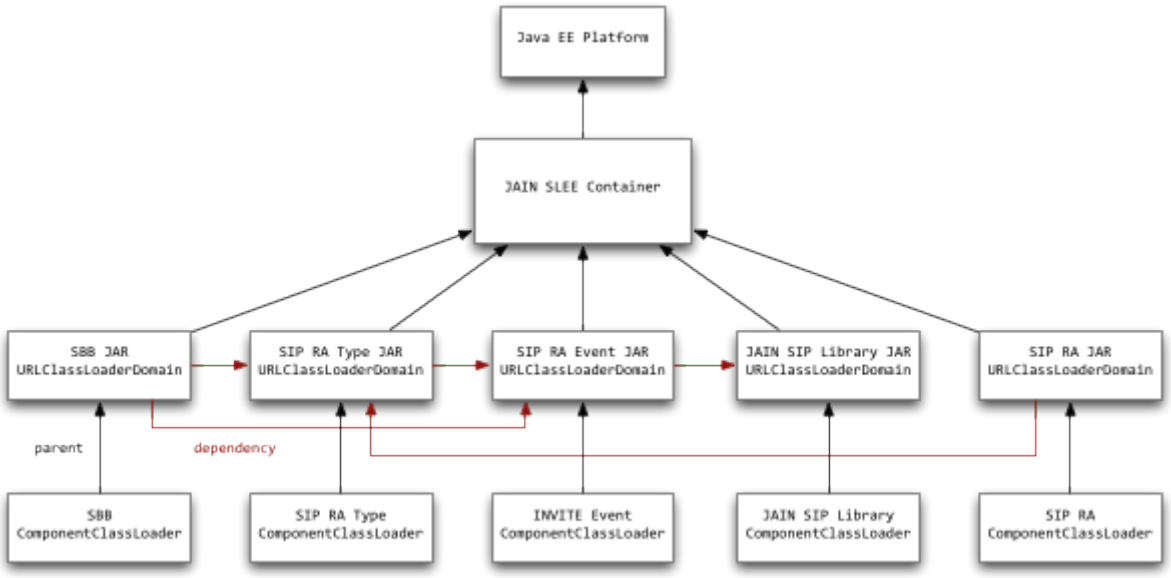

Figure 1. Classloading example in Restcomm JAIN SLEE

- The `SIP INVITE Event` component refers to the `JAIN SIP Library` in its XML descriptor, and its classloader domain depends on the classloader domain of the JAIN SIP Library.
- The `SIP RA Type` component refers to all Events in the SIP RA Event jar in its XML descriptor, and its classloader domain depends on the classloader domain of the SIP Event JAR and inherits its dependencies, including the JAIN SIP library classloading domain.
- The `SIP RA` component refers to the `SIP RA Type` component in its XML descriptor, and its classloader domain depends on the classloader domain of the SIP RA Type Component jar, and inherits its dependencies. This includes the SIP Event jar and JAIN SIP library classloading domains.
- The `SBB` component refers to the `SIP RA Type` component and `SIP INVITE Event` in its XML descriptor. Its classloader domain depends on the class loader domain of the SIP RA Type Component jar, and inherits its dependencies; the SIP Event jar and the JAIN SIP library classloading domains. It also depends on the classloader domain of the SIP Event jar.

JBoss Application Server does not see the classes of deployed JAIN SLEE components. This means that if it exports its classes for components that are complemented with Java EE components, the common classes must be deployed on JBoss Application Server , either directly or included in the Java EE component.

Congestion Control

JAIN SLEE can monitor the memory available in the JVM. In case it drops to a certain level (percentage), new events and/or activity startups are rejected, and at the same time a JAIN SLEE Alarm (which can send JMX notifications) is raised. This feature is called Congestion Control, and the container will turn it off automatically once another available memory level is reached.

If Congestion Control rejects an operation, a `javax.slee.SleeException` is thrown. This means that if the feature is to be used, the Resource Adaptors and Applications need to handle such use case, and behave properly.

The type of JAIN SLEE Alarm raised is `org.mobicents.slee.management.alarm.congestion`.

Congestion Control is turned off by default.

Congestion Control Configuration

The Congestion Control feature is configured through an XML file or through a JMX MBean. Changes applied through JMX are not persisted, and once the container is restarted the configuration will revert to the one in the XML file.

Congestion Control Persistent Configuration

Configuration is done through a XML descriptor for each [_server_profiles]. The XML file is named *jboss-beans.xml* and is located at *$JBOSS_HOME/server/profile_name/deploy/restcomm-slee/META-INF*, where `profile_name` is the server profile name.

The configuration is exposed a JBoss Microcontainer Bean:

```
<bean name="Mobicents.JAINSLEE.CongestionControlConfiguration"
    class=
"org.mobicents.slee.container.management.jmx.CongestionControlConfiguration">
    <annotation>@org.jboss.aop.microcontainer.aspects.jmx.JMX(name=
        "org.mobicents.slee:name=CongestionControlConfiguration",exposedInterface=

org.mobicents.slee.container.management.jmx.CongestionControlConfigurationMBean.class,
        registerDirectly=true)</annotation>
    <property name="periodBetweenChecks">0</property>
    <property name="minFreeMemoryToTurnOn">10</property>
    <property name="minFreeMemoryToTurnOff">20</property>
    <property name="refuseStartActivity">true</property>
    <property name="refuseFireEvent">false</property>
</bean>
```

Table 1. JAIN SLEE Congestion Control Bean Configuration

Property Name	Property Type	Description
periodBetweenChecks	int	The available memory level is checked periodically, this property defines the period, in seconds, between these checks, and if set to 0 turns off the Congestion Control feature.
minFreeMemoryToTurnOn	int	This property defines the minimum free memory percentage, which if reached turns ON the Congestion Control feature.
minFreeMemoryToTurnOff	int	This property defines the minimum free memory percentage, which if reached turns OFF the Congestion Control feature. This value should be considerably higher than minFreeMemoryToTurnOn, otherwise the feature may be turning on and off all the time.
refuseStartActivity	boolean	If true and the Congestion Control feature is ON, the container rejects activity startups, no matter it's a request from a Resource Adaptor or SBB.

Property Name	Property Type	Description
refuseFireEvent	boolean	If true and the Congestion Control feature is ON, the container rejects the firing of events, no matter it's a request from a Resource Adaptor or SBB.

Congestion Control JMX Configuration

Through JMX, the Congestion Control feature configuration can be changed with the container running. These configuration changes are not persisted.

The JMX MBean which can be used to change the Congestion Control configuration is named `org.mobicents.slee:name=CongestionControlConfiguration`, and provides getters and setters to change each property defined in the persistent configuration. The JMX Console can be used to use this MBean, see [_management_jmx_console].

JAIN SLEE 1.1 Profiles JPA Mapping

As mentioned in the containers configuration section, Restcomm JAIN SLEE uses JPA to store all JAIN SLEE 1.1 Profiles, and in mentioned section it was explained how to define which JPA / Hibernate data source. In this section more details are provided about how JAIN SLEE 1.1 Profiles are mapped to a JPA datasource schema.

Profile Specification JPA Tables And columns

For each Profile Specification, at least one Table is created, and is named `SLEE_PE_` concatenated with the Profile CMP interface simple name (`obtained as java.lang.Class.getSimpleName()`), then `_`, and finally the absolute value of the `hashCode()` method of the javax.slee.ComponentID of the Profile Specification.

This table has a primary key composed by the profile name and profile table name, and a column for each attribute of the Profile Specification CMP, except for those of array type. Those columns are named `C`, concatenated with the cmp attribute name.

For each Profile CMP attribute of `array` type, a join table is created, and is named `SLEE_PEAAV_` concatenated with the Profile CMP interface simple name (`obtained as java.lang.Class.getSimpleName()`), then `_`, then the absolute value of `hashCode()` method of the javax.slee.ComponentID of the Profile Specification, and finally the CMP attribute name. This table has a generated primary key column named `ID`, the foreign key, and two columns to store the CMP attribute value:

`SERIALIZABLE`
: Used to store the value if its type does not allow it to be converted to a String.

`STRING`
: Used when the CMP attribute type can be converted to a `java.lang.String`, for instance an

`Integer`.

Profile Specification JPA Datasource

Unless configured manually, Restcomm JAIN SLEE uses the default datasource of &JEE.PLATFORM;. Please refer to its documentation to learn about it.

Testing the JAIN SLEE 1.1 TCK

To run the JAIN SLEE 1.1 TCK:

1. Checkout and build the container source code as explained in [_install].
2. Install the TCK Resource Adaptor and Plugin:

```
cd tck/jain-slee-1.1
mvn install
```

3. Setup JAVA_OPTS environment variable to include -XX:MaxPermSize=128M -Djboss.defined.home=`/Volumes/2/jboss-5.1.0.GA` -Djava.security.manager=default -Djava.security.policy=`file:///Volumes/2/workspace/restcomm-jainslee-2/`tck/jain-slee-1.1/tck-security.policy, replacing the `absolute paths` with $JBOSS_HOME and the path of the Git checkout.
4. Unzip and run the JAIN SLEE 1.1 TCK distribution:

```
unzip testsuite.zip
cd testsuite
ant
```

No test should fail.

Setting JAIN SLEE Source Code Projects in Eclipse IDE

The JAIN SLEE Core, each RA, and each example, are worked out with separated Eclipse IDE Projects.

There are two alternatives to set up a specific project:

Procedure: Via Command Line

1. In the checked out directory of the project, and with Eclipse IDE closed, open a terminal.
2. Run the following:

```
]mvn restcomm:eclipse
mvn -Declipse.workspace=YOUR_RELATIVE_PATH_TO_ECLIPSE_WORKSPACE eclipse:add-maven-
repo
```

3. Install M2Eclipse if you want to do maven builds within Eclipse.

Procedure: With Eclipse IDE

1. Install the M2Eclipse plugin and use "Import..." and subselect the "Maven Projects" feature.

Ensure the "Resolve Workspace projects" and "Separate projects for modules" in the "Advanced" options on the bottom of the window are turned off. If the project is large, such as the JAIN SLEE Core, M2Eclipse may be a considerable slower option, due to dynamic Maven2 Dependency Management.