



Leading Open Source Middleware

# Administration Guide

JOnAS Team ( Adriana Danes)

- March 2009 -

Copyright © OW2 Consortium 2008-2009

This work is licensed under the Creative Commons Attribution-ShareAlike License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/2.0/deed.en> or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

---

# Table of Contents

1. Introduction .....	1
1.1. Terminology .....	1
1.1.1. Server or JOnAS instance .....	1
1.1.2. Service .....	1
1.1.3. Container .....	1
1.1.4. Domain .....	1
1.1.5. Master server .....	1
1.1.6. Cluster .....	1
1.2. Administration targets .....	2
1.3. Administration tools .....	2
1.4. Administration functions .....	3
1.4.1. Server administration .....	3
1.4.2. Domain administration .....	3
1.4.3. Cluster administration .....	3
2. JonasAdmin .....	4
2.1. Installing JonasAdmin .....	4
2.2. Using JonasAdmin .....	4
2.2.1. Running JonasAdmin .....	4
2.2.2. Server management .....	6
2.2.3. Note regarding persistent reconfiguration facilities .....	7
2.2.4. Domain management .....	7
3. The administration components .....	9
3.1. Management EJB Component .....	9
3.1.1. Access to the Management EJB Component .....	9
3.2. The JOnAS MBeans .....	9
3.2.1. Standard MBean .....	10
3.2.2. Other MBeans MBeans .....	10
3.3. JOnAS thread dump feature .....	10
3.3.1. Available operations .....	10
3.3.2. JonasAdmin visualization module .....	11
4. JMX Remote connectors support .....	14
4.1. JSR 160 support in JOnAS .....	14
4.1.1. Target Audience and Rationale .....	14
4.1.2. What is JSR 160 ? .....	14
4.1.3. Connector servers created by JOnAS .....	14

---

# Chapter 1. Introduction

## 1.1. Terminology

### 1.1.1. Server or JOnAS instance

A server, or JOnAS instance, is a java process started via the *jonas start* command, or via the administration tool Java EE.

Several servers may run on the same physical host.

### 1.1.2. Service

When a server starts, services are started.

A service typically provides system resources to containers. Most of the components of the JOnAS application server are pre-defined services. However, it is possible and easy for an advanced user to define a new service and to integrate it into JOnAS.

JOnAS services are manageable through JMX.

### 1.1.3. Container

A container consists of a set of Java classes that implement the Java EE specification. The role of the container is to provide the facilities for executing Java EE components.

There are three types of containers:

- EJB container in which Enterprise JavaBeans are deployed and run
- Web container for JSPs and servlets
- Client container

### 1.1.4. Domain

A domain represents an administration perimeter which is under the control of an administration authority.

This perimeter contains management targets like servers and clusters.

If a domain contains several elements, it provides at least one common administration point represented by a master server.

### 1.1.5. Master server

A master is a JOnAS instance having particular management capabilities within the domain:

- it is aware of the domain's topology
- it allows management and monitoring of all the elements belonging to the domain

### 1.1.6. Cluster

A cluster is a group of JOnAS servers having common properties within a domain. It usually allows to run a J2EE application, or a J2EE module, on the cluster members as if they were a single server. The objective is to achieve applications scalability and high availability.

JOnAS supports several cluster types:

- Clusters for Web level load-balancing
- Clusters for high availability of Web components
- Clusters for EJB level load-balancing
- Clusters for high availability of EJB components
- Clusters for JMS destination scalability and high availability
- Clusters for administration purpose which facilitate management operations like deployment / undeployment.

From the administrator point of view, a cluster represents a single administration target.

Note that a particular JOnAS server may belong to zero, one or more clusters.

## 1.2. Administration targets

This section presents the possible targets of a JOnAS administrator. An administrator may act on the following targets:

- An individual JOnAS server.
- A management domain.
- A management cluster.

The servers composing a management domain can be grouped into one or more clusters. These are clustered servers. A domain may have clustered and un-clustered servers.

Each target is identified by a unique name.

- A JOnAS server name is given by the value of the `-n` option used in the `jonas start` command (the default name is `jonas`).
- The domain name is given by the value of the `domain.name` environment variable (by default, a server is started in a domain having the server's name). See [How to configure a domain \[configuration\\_guide.html#clustering.configDomain\]](#) for more details.
- The naming of the clusters depend on configuration parameters which are different for the different cluster types.

## 1.3. Administration tools

This section presents the means offered to a JOnAS administrator to act on an administration target:

- The JOnAS commands, particularly the `jonas admin` [[command\\_guide.html#commands.jonas.jonasadmin](#)] command.
- Management applications deployed on a JOnAS server. This may be the `JonasAdmin` web application, or any custom J2EE management application based on the Section 3.1, “Management EJB Component”.
- Any generic JMX management console connected to a target server via a JMX Remote Connector Section 4.1, “JSR 160 support in JOnAS”.
- WebService clients using the WebService end-point exposed by the Management EJB

- Other mechanisms like ant tasks.

## 1.4. Administration functions

This section presents the operations provided to the JOnAS administrator.

These operations depend on the management target.

### 1.4.1. Server administration

JOnAS administration mainly provides monitoring and configuration functions. Moreover, administrator can dynamically install/deploy/undeploy applications and/or resources on a running server.

All the functions are proposed by the JonasAdmin application, whereas the `jonas admin [command_guide.html#commands.jonas.jonasadmin]` command supports only a part of these functions.

### 1.4.2. Domain administration

Domain administration is mainly based on JonasAdmin.

It allows consulting the domain map (shows the servers and clusters in the domain) and modify this map.

Moreover, monitoring of the domain is provided: shows state evolution of servers and clusters, and statistics on resources used by the different servers in the domain.

Finally, dynamic install/deploy/undeploy operations are extended at the domain level: the applications source is the administrator server and operation's target may be one or more servers and/or clusters in the domain.

### 1.4.3. Cluster administration

Administrator may create clusters, add and/or remove servers to/from clusters. Allowed operations depend on the cluster type.

---

# Chapter 2. JonasAdmin

JonasAdmin is an administration tool providing a user friendly graphical console.

JonasAdmin was developed using the Struts framework; it uses standard technologies such as Java Servlets and JavaServer Pages and JMX.

## 2.1. Installing JonasAdmin

As with any web application, JonasAdmin requires a servlet server to be installed. Make sure that the web service is listed in the `jonas.services` property in the `jonas.properties` configuration file.

Designed as a web application, JonasAdmin is packed in a WAR and installed under the `JONAS_BASE/ deploy/` directory.

Once the server started, JonasAdmin is automatically deployed if the **depmoitor** service is activated. Otherwise, JonasAdmin can be deployed manually using the `jonas admin [command_guide.html#commands.jonas.jonasadmin]` command.

When accessing JonasAdmin, the administrator must provide identification and authentication. The `jonas-realm.xml` configuration file contains a memory realm definition named `memrlm_1`, which is referenced in both `tomcat6-server.xml` (for Tomcat) and `jetty.xml` (for Jetty) configuration files. The default user name (`jonas`) and password (`jonas`) corresponding to the `admin` role can be modified here.

## 2.2. Using JonasAdmin

Once started, JonasAdmin can administer the JOnAS server in which it is running. If the host server is a master, JonasAdmin can also administer the rest of the servers running in the domain.

### 2.2.1. Running JonasAdmin

Once the JonasAdmin application deployed on a server, the administration console is accessible at the URL: `http://<hostname>:<portnumber>/jonasAdmin/` using any web browser.

`<hostname>` is the name of the host where the Servlet container is running and `<portnumber>` is the http port number (default is 9000).

After logging in, the left-hand frame in the Welcome page displays the management tree associated with the JOnAS server running JonasAdmin. The management tree's root is `Domain`, which allows to the domain management facilities in case of a master server.


In the image below, JonasAdmin is running on the master server named **jonas** within a domain also named **jonas**. It is immediately apparent that this is a master server, as we have a `Monitoring` and a `Deployment` sub-tree under the `Domain` root node.

**JOnAS Administration - Iceweasel**

File Edit View History Bookmarks Tools Help

**Domain (jonas)**

- Monitoring
  - jonas
  - j1
- G1 - (CmiCluster)
  - jonas
  - j1
  - cd1
- Deployment
  - Applications (EAR)
  - EJB Modules (JAR)
  - Web Modules (WAR)
- Server JOnAS (jonas)
  - Monitoring
  - Logging
  - Web Protocols
  - Services
  - Deployment
    - Applications (EAR)
    - EJB Modules (JAR)
    - Web Modules (WAR)
    - All (JAR, WAR, RAR, EAR)
  - Resources
  - Security
  - Jonas MQ Connect
  - MBeans

 Domain (jonas)

**Domain**

**Domain properties**

Name
Master server

**Servers**

**Action**

[Add server](#) [Remove server](#)

<a href="#">j1</a>
<a href="#">jonas</a>

**Clusters**

**Action**

[Create cluster](#)

[G1](#)

**Cluster Daemons**

[cd1](#)

The management tree in this figure allows access to the following main management facilities:

- Domain administration with domain level deployment facilities.
- Current server administration
  - Server monitoring
  - Logging management
  - Communication protocols management
  - Active services presentation and configuration
  - Dynamic deployment at the current server level
  - Resources management
  - Security management
- Joram platform administration
- MBeans browsing

## 2.2.2. Server management

Displays general information about the administered JOnAS server, including the JMX server and the WEB server, and provides the capability of listing the content of the Registry.

### 2.2.2.1. Server monitoring

Presents memory usage, a count of the threads created by JOnAS, and other monitoring information concerning managed services and resources.

### 2.2.2.2. Logging management

Allows the administrator to configure the JOnAS Logging system. Additionally, if Tomcat is used as the WEB container service implementation, it allows creation of new access log valves.

### 2.2.2.3. Communication protocols management

This management facility relates to the integration of Tomcat management in JonasAdmin. It currently presents connectors defined in the Tomcat configuration and allows for the creation of new HTTP, HTTPS, or AJP connectors. Note that the `protocols` sub-tree is not presented if Jetty is used as the WEB container service implementation.

### 2.2.2.4. Active services presentation and configuration

All the active services have a corresponding sub-tree in the `Services` tree.

Managing the various container services consists of presenting information about the components deployed in these containers. New components can be deployed using the dynamic deployment facilities presented in the next section.

Similarly, the services that allow management of the different types of resources (DataSources, Resource Adapters and Mail resources) also provide information about the resources being deployed. Additionally, deployed resources (DataSources or MailFactories) can be reconfigured and their new configuration made persistent by using a `Save` button.

The transaction service management allows reconfiguration (possibly persistent) and presents monitoring information about transactions managed by JOnAS.



### 2.2.2.5. Dynamic deployment with JonasAdmin

A very useful management operation is the capability of installing (upload), deploying, undeploying or removing stand-alone J2EE components (JAR, WAR, RAR packages) or J2EE applications (EAR packages) in the administered server using the `Deployment` sub-tree.

The administrator's task is facilitated by the display of the list of deployable modules, the list of deployed modules, and the capability of transferring modules from one list to another (which corresponds to `deploy/undeploy` operations).

The deployable modules can be installed in directories specific to their type, for example, the deployable JARs are un-deployed JARs installed in `JONAS_BASE/ejbjars/` directory, or in the `JONAS_BASE/deploy` directory.

The `Deployment` sub-tree also allows a J2EE package to be uploaded from the local file system to the corresponding directory of the administered server (`install` operation), and the opposite `remove` operation.

### 2.2.2.6. Resources management

The `Resources` sub-tree provides the capability of loading or creating new resources managed by the active services.

### 2.2.2.7. Security management

The `Security` sub-tree presents existing security realms and allows the creation of new realms of different types: memory, datasource, and ldap realms.

## 2.2.3. Note regarding persistent reconfiguration facilities

It is important to note that JOnAS and Tomcat have different approaches to reconfiguration persistency. In JOnAS, every `Save` operation is related to a service or a resource reconfiguration. For example, the administrator can reconfigure a service and a resource, but choose to save only the new resource configuration. In Tomcat, the `Save` operation is global to all configuration changes that have been performed. For example, if a new HTTP connector is reconfigured and a new context created for a web application, both configuration changes are saved when using the `Save` button.

## 2.2.4. Domain management

First recall that domain management functions are accessible only when JonasAdmin is deployed on a master server. The `Domain` tree contains only one `Server` sub-tree, the currently administered server, which is initially the server hosting JonasAdmin.

Domain management principal function is to present the domain topology: list all the servers and clusters belonging to the domain. It also allows modification of the domain topology by adding new servers and clusters to the domain, removing servers and moving servers to/from clusters.

The domain management page also presents servers that are not yet started but are specified as belonging to the domain in the new configuration file named `domain.xml`. Also, a server can be added to the domain when it has been started without having the discovery service enabled.

An essential domain management function is that the administrator can switch from the master to any of the other servers in the domain. Currently, JonasAdmin allows only one category of global domain level management operation, the `deployment` operation. Using any other management operation requires switching to the server to be administered.

Domain level deployment allows for `deploying` one or more J2EE packages (JARs, WARs, RARs or EARs), which are installed in the corresponding master directory (`ejbjars`, `webaps`, `rars` or `apps`), into any running server in the domain. A `deployment` operation target may be a server but also a cluster. The `deploy` operation may have three semantics:

- `deploy only` (create container) - which is useful when the package is already installed on the target server.
- `distribute only` - which means install the package in the target's corresponding directory
- `distribute and (re)deploy` the package, with optionally replacing the current package with the new one.

Note that at domain level deployment the `Upload` and `Remove` operations are only related to the master server itself.

---

# Chapter 3. The administration components

## 3.1. Management EJB Component

The MEJB is specified by the J2EE Management Specification which defines the J2EE Management Model.

The MEJB component exposes the managed objects within the JOnAS platform as JMX manageable resources (MBeans). It is packed in an ejb-jar file installed in the JONAS\_ROOT/deploy directory, and therefore it can be loaded at server start-up.

The MEJB component is registered in JNDI under the name `ejb/mgmt/MEJB`.

### 3.1.1. Access to the Management EJB Component

Access to the MEJB provided by the JOnAS 5 distribution is now role secured. The client application that wants to access it must have one of the following roles:

`mejb-user` for accessing to the read-only operation of the MEJB

`mejb-admin` role allowing access for all operation of the MEJB

Unauthorized access is forbidden

This snippet includes all of the elements needed to specify security roles in the `web.xml` file of a web application client of MEJB

```
<security-constraint>
<web-resource-collection>
  <web-resource-name>Protected Area</web-resource-name>
  <!-- Define the context-relative URL(s) to be protected -->
  <url-pattern>/*</url-pattern>
  <!-- If you list http methods, only those methods are protected -->
  <http-method>DELETE</http-method>
  <http-method>GET</http-method>
  <http-method>POST</http-method>
  <http-method>PUT</http-method>
</web-resource-collection>
<auth-constraint>
  <!-- Anyone with one of the listed roles may access this area -->
  <role-name>mejb-admin</role-name>
</auth-constraint>
</security-constraint>

<!-- Default login configuration uses BASIC authentication -->
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>JOnAS Realm</realm-name>
</login-config>

<!-- Security roles referenced by this web application -->
<security-role>
  <role-name>mejb-admin</role-name>
</security-role>
```

## 3.2. The JOnAS MBeans

JOnAS administration is based on the standard MBeans defined by the J2EE Management Specification, and on more MBeans provided by JOnAS itself and by components integrated in JOnAS: Tomcat/Jetty server, JORAM platform, JMX Server, etc.

## 3.2.1. Standard MBean

All the MBeans defined by the J2EE Management Specification are provided by JOnAS. These MBeans respect the naming conventions and are registered in the JMX server embeded in any JOnAS instance.

### 3.2.1.1. J2EEServer

The J2EEServer MBean provides attributes and methods for the server management. For example, it supports resource and application deployment. It also provides server configuration and monitoring information.

### 3.2.1.2. J2EEDomain

The J2EEDomain MBean provides domain management operation support. Most of the exposed operations are usefull only in the case of a master instance.

## 3.2.2. Other MBeans MBeans

Proprietary MBeans are registered by JOnAS and by the integrated components in order to allow management of services or specific resources.

## 3.3. JOnAS thread dump feature

The aim of JOnAS thread dump feature is to provide an access to a thread stack dump from the underlying JVM of a running JOnAS server for monitoring purpose. The thread dump access is done through a JMX connection to the JOnAS "J2EEServer" MBean (named such as "\$domainname:j2eeType=J2EEServer,name=\$jonasname" depending on the JOnAS domain and server name). This feature gathers few operations directly available on JOnAS server through a JMX connection to the "J2EEServer" MBean " or through a Flex visualization module integrated in the web-based JonasAdmin application.

### 3.3.1. Available operations

A set of operations is available on the remote server, it can be accessed by any JMX client connecting to the JOnAS J2EEServer Mbean. Here is a list of the available operations with an example using the JConsole:

- **getThreadStackDump**

This operation gives the JVM thread stack dump as a string. The result of this operation is the same as reading the attribute **threadStackDump**.

- **logThreadStackDump**

This operation logs the JVM thread stack dump in JOnAS log file (level:info).

- **printThreadStackDump**

This operation prints the JVM thread stack dump in a given file on the remote server.

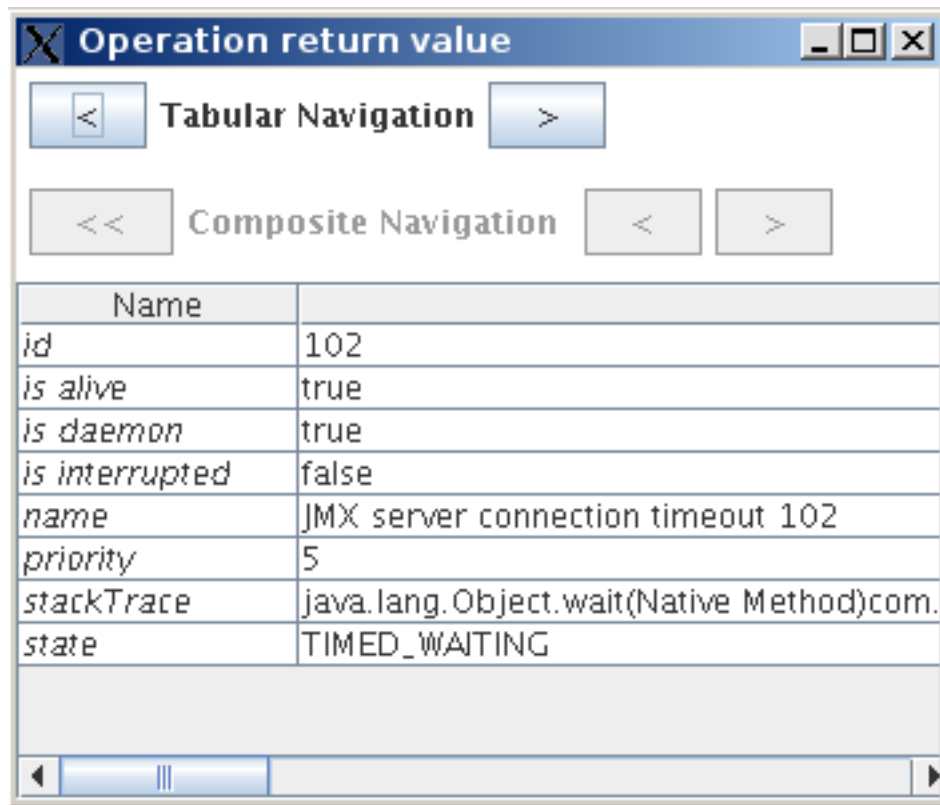
- **getThreadStackDumpList**

This operation gives the JVM thread stack dump as a TabularData object. The composition of the returned object is a set of properties-values elements, each element representing a givean thread. Thus, such a structured object can be easily used in a script.

For instance, the above operations can be easily tried using the JConsole displaying the following interface:



Here is a result of the `getThreadStackDumpList` operation given by the JConsole. Navigation between elements is done using right and left arrows, each thread properties are displayed.



Please note all the above operations are also available through the MBeanCmd [[http://jasmine.ow2.org/doc/current/mbeancommand\\_guide/index.php](http://jasmine.ow2.org/doc/current/mbeancommand_guide/index.php)] Dump Command

### 3.3.2. JonasAdmin visualization module

JonasAdmin web application provides a Flex module that lets you view all threads information. This module is available in the "Monitoring" node from the "Server JOnAS" section by clicking on the "StackDump" tab.

The screenshot shows the JonasAdmin web console interface. On the left is a navigation tree for the 'jonas' domain. The right pane shows the 'Monitoring' page for 'Server JOnAS (jonas)', with tabs for 'General', 'Memory', and 'StackDump'. The 'Threads' section displays a table of active threads.

**Navigation Tree:**

- JonasAdmin
  - Domain/Flex (jonas)
  - Domain Deployment
  - Domain (jonas)
  - Server JOnAS (jonas)
    - Monitoring**
    - Logging
    - Web Protocols
    - Services
    - Deployment
      - Applications (EAR)
      - EJB Modules (JAR)
      - Web Modules (WAR)
      - Resource Adapter Modules (RAR)
      - All (JAR, WAR, RAR, EAR)
    - Resources
    - Security
    - Jonas MQ Connect
    - Joram platform
    - MBeans

**Monitoring Page:**

JonasAdmin > Server JOnAS (jonas) > Monitoring

General | **Memory** | StackDump

Threads :

id ▲	name
14	FelixPackageAdmin
15	Thread-3
16	Configuration Updat
19	RMI TCP Accept-109
22	JonasBatch
23	JonasClock
25	RMI Reaper
26	GC Daemon
29	RMI RenewClean-[12
32	RMI LeaseChecker
37	JMX server connectio
38	Thread-8
39	DomainStateMonitor
40	Thread-9
41	JotmBatch
42	JotmClock

Buttons: Refresh, Select all

If JOnAs is started as a master ("jonas.master" property in the "jonas.properties" file is set to "true"), the thread stack dump visualization module is available for the current administrated server. Thus a master can access the thread stack dump of any server from the same domain through jonasAdmin stackDump visualization module by switching the current administrated server.

The visualization module shows all threads name and id from the JVM. Threads can be ordered by names or ids by clicking on the header of the related column. A selection of one or more threads can be done by clicking on names or ids. Selected threads information such as priority, state or stacktrace appear in a column to the right.

The **Refresh** button refreshes the threads list from the underlying JVM.

The **Select all** button allows to select all the threads from the JVM to get the complete "stacktrace" information from the server.

The **Copy to clipboard** button allows to copy all selected threads information in the clipboard so they could be pasted in a file to be kept. (Please also note that any part from the right column can be selected using the mouse and copied with a right click).

---

# Chapter 4. JMX Remote connectors support

## 4.1. JSR 160 support in JOnAS

### 4.1.1. Target Audience and Rationale

JOnAS provides support for remote connection to the MBean server in a standard way based on the JMX Remote API.

### 4.1.2. What is JSR 160 ?

The JSR 160 specifies the JMX Remote API which extends the JMX specification by providing a standard way to connect to remote JMX-enabled applications.

Currently, JSR 160 has defined a mandatory connector based on RMI (that supports both RMI/JRMP and RMI/IIOP).

### 4.1.3. Connector servers created by JOnAS

JSR 160 support implies providing standard connector server objects. The JMX service creates at start-up one or several such objects, depending on the protocols configuration (defined in `carol.properties` file). To create a client connector, the client side needs to know the URL of the connector server. Below we present the URLs that can be used by the clients depending on the protocol they choose.

Currently only 2 protocols can be used by JSR-160 connectors: RMI/JRMP and RMI/IIOP.

#### 4.1.3.1. Using a RMI/JRMP Connector

This connector can be used if the `jrmp` protocol is set in the `carol.protocols` list.

The client has to construct a `JMXServiceURL` using the following `String`, possibly modified according to the JOnAS-specific configuration: `service:jmx:rmi:///jndi/rmi://host:port/jrmpconnector_jonasServerName` where `host` is the host on which is running the JOnAS server to be managed. The `port` number is given in the `carol.properties` file.

Then, a `JMXConnector` has to be created and connected to the connector server using the JMX Remote API.

#### 4.1.3.2. Example 1:

```
Hashtable environment = null;
JMXServiceURL address = new JMXServiceURL("service:jmx:rmi:///jndi/rmi://host:1099/
jrmpconnector_jonas");
JMXConnector connector = JMXConnectorFactory.newJMXConnector(address, environment);
connector.connect(environment);
```

#### 4.1.3.3. Using a RMI/IIOP Connector

This connector can be used if the `iiop` protocol is set in the `carol.protocols` list.



The client code is similar to the JRMP case, but the `String` to be used to construct the `JMXServiceURL` must adhere to the following model: `"service:jmx:iiop:///jndi/iiop://host:port/iiopconnector_ jonasServerName"`

#### 4.1.3.4. Example 2:

```
Hashtable environment = null;
JMXServiceURL address = new JMXServiceURL("service:jmx:iiop:///jndi/iiop://host:2001/
iiopconnector_jonas");
JMXConnector connector = JMXConnectorFactory.newJMXConnector(address, environment);
connector.connect(environment);
```