| | [**Overview**](http://docs.google.com/overview-summary.html) | **Package** | Class | [**Use**](http://docs.google.com/package-use.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV PACKAGE**](http://docs.google.com/javax/management/remote/package-summary.html)   [**NEXT PACKAGE**](http://docs.google.com/javax/management/timer/package-summary.html) | [**FRAMES**](http://docs.google.com/index.html?javax/management/remote/rmi/package-summary.html)    [**NO FRAMES**](http://docs.google.com/package-summary.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |

## Package javax.management.remote.rmi

The RMI connector is a connector for the JMX Remote API that uses RMI to transmit client requests to a remote MBean server.

**See:**

[**Description**](#3znysh7)

| **Interface Summary** | |
| --- | --- |
| [**RMIConnection**](http://docs.google.com/javax/management/remote/rmi/RMIConnection.html) | RMI object used to forward an MBeanServer request from a client to its MBeanServer implementation on the server side. |
| [**RMIServer**](http://docs.google.com/javax/management/remote/rmi/RMIServer.html) | RMI object used to establish connections to an RMI connector. |

| **Class Summary** | |
| --- | --- |
| [**RMIConnectionImpl**](http://docs.google.com/javax/management/remote/rmi/RMIConnectionImpl.html) | Implementation of the [RMIConnection](http://docs.google.com/javax/management/remote/rmi/RMIConnection.html) interface. |
| [**RMIConnectionImpl\_Stub**](http://docs.google.com/javax/management/remote/rmi/RMIConnectionImpl_Stub.html) |  |
| [**RMIConnector**](http://docs.google.com/javax/management/remote/rmi/RMIConnector.html) | A connection to a remote RMI connector. |
| [**RMIConnectorServer**](http://docs.google.com/javax/management/remote/rmi/RMIConnectorServer.html) | A JMX API connector server that creates RMI-based connections from remote clients. |
| [**RMIIIOPServerImpl**](http://docs.google.com/javax/management/remote/rmi/RMIIIOPServerImpl.html) | An [RMIServerImpl](http://docs.google.com/javax/management/remote/rmi/RMIServerImpl.html) that is exported through IIOP and that creates client connections as RMI objects exported through IIOP. |
| [**RMIJRMPServerImpl**](http://docs.google.com/javax/management/remote/rmi/RMIJRMPServerImpl.html) | An [RMIServer](http://docs.google.com/javax/management/remote/rmi/RMIServer.html) object that is exported through JRMP and that creates client connections as RMI objects exported through JRMP. |
| [**RMIServerImpl**](http://docs.google.com/javax/management/remote/rmi/RMIServerImpl.html) | An RMI object representing a connector server. |
| [**RMIServerImpl\_Stub**](http://docs.google.com/javax/management/remote/rmi/RMIServerImpl_Stub.html) |  |

## Package javax.management.remote.rmi Description

The RMI connector is a connector for the JMX Remote API that uses RMI to transmit client requests to a remote MBean server. This package defines the classes that the user of an RMI connector needs to reference directly, for both the client and server sides. It also defines certain classes that the user will not usually reference directly, but that must be defined so that different implementations of the RMI connector can interoperate.

The RMI connector supports both the JRMP and the IIOP transports for RMI.

Like most connectors in the JMX Remote API, an RMI connector usually has an address, which is a [JMXServiceURL](http://docs.google.com/javax/management/remote/JMXServiceURL.html). The protocol part of this address is rmi for a connector that uses the default RMI transport (JRMP), or iiop for a connector that uses RMI/IIOP.

There are two forms for RMI connector addresses:

* In the *JNDI form*, the URL indicates *where to find an RMI stub for the connector*. This RMI stub is a Java object of type [RMIServer](http://docs.google.com/javax/management/remote/rmi/RMIServer.html) that gives remote access to the connector server. With this address form, the RMI stub is obtained from an external directory entry included in the URL. An external directory is any directory recognized by [JNDI](http://docs.google.com/javax/naming/package-summary.html), typically the RMI registry, LDAP, or COS Naming.
* In the *encoded form*, the URL directly includes the information needed to connect to the connector server. When using RMI/JRMP, the encoded form is the serialized RMI stub for the server object, encoded using BASE64 without embedded newlines. When using RMI/IIOP, the encoded form is the CORBA IOR for the server object.

Addresses are covered in more detail below.

### Creating an RMI connector server

The usual way to create an RMI connector server is to supply an RMI connector address to the method [JMXConnectorServerFactory.newJMXConnectorServer](http://docs.google.com/javax/management/remote/JMXConnectorServerFactory.html#newJMXConnectorServer(javax.management.remote.JMXServiceURL,%20java.util.Map,%20javax.management.MBeanServer)). The MBean server to which the connector server is attached can be specified as a parameter to that method. Alternatively, the connector server can be registered as an MBean in that MBean server.

An RMI connector server can also be created by constructing an instance of [RMIConnectorServer](http://docs.google.com/javax/management/remote/rmi/RMIConnectorServer.html), explicitly or through the MBean server's createMBean method.

#### Choosing the RMI transport

You can choose the RMI transport (JRMP or IIOP) by specifying rmi or iiop in the *protocol* part of the serviceURL when creating the connector server. You can also create specialised connector servers by instantiating an appropriate subclass of [RMIServerImpl](http://docs.google.com/javax/management/remote/rmi/RMIServerImpl.html) and supplying it to the RMIConnectorServer constructor.

#### Connector addresses generated by the server

If the serviceURL you specify has an empty URL path (after the optional host and port), or if you do not specify a serviceURL, then the connector server will fabricate a new JMXServiceURL that clients can use to connect:

* If the serviceURL looks like:  
   service:jmx:rmi://*host*:*port*  
     
  then the connector server will generate an [RMIJRMPServerImpl](http://docs.google.com/javax/management/remote/rmi/RMIJRMPServerImpl.html) and the returned JMXServiceURL looks like:  
   service:jmx:rmi://*host*:*port*/stub/*XXXX*  
     
  where *XXXX* is the serialized form of the stub for the generated object, encoded in BASE64 without newlines.
* If the serviceURL looks like:  
   service:jmx:iiop://*host*:*port*  
     
  then the connector server will generate an [RMIIIOPServerImpl](http://docs.google.com/javax/management/remote/rmi/RMIIIOPServerImpl.html) and the returned JMXServiceURL looks like:  
   service:jmx:iiop://*host*:*port*/ior/IOR:*XXXX*  
     
  where IOR:*XXXX* is the standard CORBA encoding of the Interoperable Object Reference for the generated object.
* If there is no serviceURL, there must be a user-provided RMIServerImpl. If the [toStub](http://docs.google.com/javax/management/remote/rmi/RMIServerImpl.html#toStub()) method on this object returns an instance of [Stub](http://docs.google.com/javax/rmi/CORBA/Stub.html), then the connector server will generate a JMXServiceURL using the iiop form above. Otherwise, it will generate a JMXServiceURL using the rmi form.

The *host* in a user-provided serviceURL is optional. If present, it is copied into the generated JMXServiceURL but otherwise ignored. If absent, the generated JXMServiceURL will have the local host name.

The *port* in a user-provided serviceURL is also optional. If present, it is also copied into the generated JMXServiceURL; otherwise, the generated JMXServiceURL has no port. For an serviceURL using the rmi protocol, the *port*, if present, indicates what port the generated remote object should be exported on. It has no other effect.

If the user provides an RMIServerImpl rather than a JMXServiceURL, then the generated JMXServiceURL will have the local host name in its *host* part and no *port*.

#### Connector addresses based on directory entries

As an alternative to the generated addresses just described, the serviceURL address supplied when creating a connector server can specify a *directory address* in which to store the provided or generated RMIServer stub. This directory address is then used by both client and server.

In this case, the serviceURL has one of these two forms:

service:jmx:rmi://*host*:*port*/jndi/*jndi-name*  
 service:jmx:iiop://*host*:*port*/jndi/*jndi-name*

Here, *jndi-name* is a string that can be supplied to [javax.naming.InitialContext.bind](http://docs.google.com/javax/naming/InitialContext.html#bind(java.lang.String,%20java.lang.Object)).

As usual, the *host* and :*port* can be omitted.

The connector server will generate an RMIServerImpl based on the protocol (rmi or iiop) and, for rmi, the *port* if any. When the connector server is started, it will derive a stub from this object using its [toStub](http://docs.google.com/javax/management/remote/rmi/RMIServerImpl.html#toStub()) method and store the object using the given *jndi-name*. The properties defined by the JNDI API are consulted as usual.

For example, if the JMXServiceURL is:

service:jmx:rmi://ignoredhost/jndi/rmi://myhost/myname

then the connector server will generate an RMIJRMPServerImpl and store its stub using the JNDI name

rmi://myhost/myname

which means entry myname in the RMI registry running on the default port of host myhost. Note that the RMI registry only allows registration from the local host. So, in this case, myhost must be the name (or a name) of the host that the connector server is running on.

In this JMXServiceURL, the first rmi: specifies the RMI connector, while the second rmi: specifies the RMI registry.

As another example, if the JMXServiceURL is:

service:jmx:iiop://ignoredhost/jndi/ldap://dirhost:9999/cn=this,ou=that

then the connector server will generate an RMIIIOPServerImpl and store its stub using the JNDI name

ldap://dirhost:9999/cn=this,ou=that

which means entry cn=this,ou=that in the LDAP directory running on port 9999 of host dirhost.

If the JMXServiceURL is:

service:jmx:iiop://ignoredhost/jndi/cn=this,ou=that

then the connector server will generate an RMIIIOPServerImpl and store its stub using the JNDI name

cn=this,ou=that

For this case to work, the JNDI API must have been configured appropriately to supply the information about what directory to use.

In these examples, the host name ignoredhost is not used by the connector server or its clients. It can be omitted, for example:

service:jmx:iiop:///jndi/cn=this,ou=that

However, it is good practice to use the name of the host where the connector server is running. This is often different from the name of the directory host.

#### Connector server attributes

When using the default JRMP transport, RMI socket factories can be specified using the attributes jmx.remote.rmi.client.socket.factory and jmx.remote.rmi.server.socket.factory in the environment given to the RMIConnectorServer constructor. The values of these attributes must be of type [RMIClientSocketFactory](http://docs.google.com/java/rmi/server/RMIClientSocketFactory.html) and [RMIServerSocketFactory](http://docs.google.com/java/rmi/server/RMIServerSocketFactory.html), respectively. These factories are used when creating the RMI objects associated with the connector.

### Creating an RMI connector client

An RMI connector client is usually constructed using [JMXConnectorFactory](http://docs.google.com/javax/management/remote/JMXConnectorFactory.html), with a JMXServiceURL that has rmi or iiop as its protocol.

If the JMXServiceURL was generated by the server, as described above under ["connector addresses generated by the server"](#_2et92p0), then the client will need to obtain it directly or indirectly from the server. Typically, the server makes the JMXServiceURL available by storing it in a file or a lookup service.

If the JMXServiceURL uses the directory syntax, as described above under ["connector addresses based on directory entries"](#_tyjcwt), then the client may obtain it as just explained, or client and server may both know the appropriate directory entry to use. For example, if the connector server for the Whatsit agent uses the entry whatsit-agent-connector in the RMI registry on host myhost, then client and server can both know that the appropriate JMXServiceURL is:

service:jmx:rmi:///jndi/rmi://myhost/whatsit-agent-connector

If you have an RMI stub of type [RMIServer](http://docs.google.com/javax/management/remote/rmi/RMIServer.html), you can construct an RMI connection directly by using the appropriate constructor of [RMIConnector](http://docs.google.com/javax/management/remote/rmi/RMIConnector.html).

### Specifying an ORB for the RMI/IIOP connector

When using the IIOP transport, the client and server can specify what ORB to use with the attribute java.naming.corba.orb. Connection to the ORB happens at [start](http://docs.google.com/javax/management/remote/rmi/RMIConnectorServer.html#start()) time for the connector server, and at [connect](http://docs.google.com/javax/management/remote/rmi/RMIConnector.html#connect(java.util.Map)) time for the connector client. If the java.naming.corba.orb attribute is contained in the environment Map, then its value (an [ORB](http://docs.google.com/org/omg/CORBA/ORB.html)), is used to connect the IIOP Stubs. Otherwise, a new org.omg.CORBA.ORB is created by calling [org.omg.CORBA.ORB.init((String[])null,(Properties)null)](http://docs.google.com/org/omg/CORBA/ORB.html). A later RMI connector client or server in the same JVM can reuse this ORB, or it can create another one in the same way.

If the java.naming.corba.orb attribute is specified and does not point to an [ORB](http://docs.google.com/org/omg/CORBA/ORB.html), then an [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) will be thrown.

The mechanism described here does not apply when the IIOP Remote objects (Stubs or Servers) are created and connected to an ORB manually before being passed to the RMIConnector and RMIConnectorServer.

### Dynamic code downloading

If an RMI connector client or server receives from its peer an instance of a class that it does not know, and if dynamic code downloading is active for the RMI connection, then the class can be downloaded from a codebase specified by the peer. The article [*Dynamic code downloading using Java RMI*](http://docs.google.com/technotes/guides/rmi/codebase.html) explains this in more detail.

**Since:** 1.5 **See Also:** [JavaTM Remote Method Invocation (RMI)](http://docs.google.com/technotes/guides/rmi/index.html),  [Java Naming and Directory InterfaceTM (JNDI)](http://docs.google.com/technotes/guides/jndi/index.html), [RFC 2045, section 6.8, "Base64 Content-Transfer-Encoding"](http://www.ietf.org/rfc/rfc2045.txt)

| | [**Overview**](http://docs.google.com/overview-summary.html) | **Package** | Class | [**Use**](http://docs.google.com/package-use.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV PACKAGE**](http://docs.google.com/javax/management/remote/package-summary.html)   [**NEXT PACKAGE**](http://docs.google.com/javax/management/timer/package-summary.html) | [**FRAMES**](http://docs.google.com/index.html?javax/management/remote/rmi/package-summary.html)    [**NO FRAMES**](http://docs.google.com/package-summary.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)

For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).