Enhancements in JDK 7

From JDK 7 Update 21, the RMI property **`java.rmi.server.useCodebaseOnly`** is set to **true** by default. In earlier releases, the default value was **`false`**.

Note: This change is also applicable to JDK 6 Update 45 and JDK 5 Update 45 releases.

When set to `false`, this property allows one side of an RMI connection to specify a network location (URL) from which the other side of the RMI connection should load Java classes. The typical use of this mechanism is for RMI clients and servers to be able to provide remote interfaces and data classes to each other, without the need for configuration.

If the JVM at one end of an RMI connection specifies one or more URLs in its **`java.rmi.server.codebase`** system property, that information is passed over the RMI connection to the other end. If the receiving JVM has its **`java.rmi.server.useCodebaseOnly`** system property set to `false`, it will then attempt to use those URLs for loading Java classes referenced in the RMI request stream.

The behavior of loading classes from locations specified by the remote end of the RMI connection, is disabled when the **`java.rmi.server.useCodebaseOnly`** is set to **`true`**. In this case, classes are loaded only from preconfigured locations, such as the locally-specified **`java.rmi.server.codebase`** property or the local CLASSPATH, and not from `codebase` information passed through the RMI request stream.

This change of default value may cause RMI-based applications to break unexpectedly. The typical symptom is a stack trace that contains a `java.rmi.UnmarshalException` containing a nested `java.lang.ClassNotFoundException`.

If these exceptions occur, the preferred way to solve the problem is to configure all RMI clients and servers to use the same codebase, by specifying proper values in the **`java.rmi.server.codebase`** system property. This is typically done by adding the `-D` option to the command that starts up the application:

```
java -Djava.rmi.server.codebase=file:///<path-to-remote-classes>/
```

It may also be necessary to adjust the permissions in the application's security policy file in order to allow access to the location specified by the URL. This involves granting permissions such as **`FilePermission`** and **`SocketPermission`**.

To configure the **rmiregistry** daemon to use a specific codebase, the following syntax can be used:

```
rmiregistry -J-Djava.rmi.server.codebase=file:///<path-to-remote-classes>/
```

A different syntax is required to specify the codebase for activation group JVMs started by **`rmid`**. This daemon does not process RMI requests itself, but it creates JVM subprocesses to handle RMI requests. The syntax for specifying the codebase for **`rmid`** subprocesses is as follows:

```
rmid -C-Djava.rmi.server.codebase=file:///<path-to-remote-classes>/
```

In some cases it may be difficult or impossible to preconfigure RMI clients or servers or the **`rmiregistry`** or **`rmid`** daemons with a specific codebase. If this is the case, a way to resolve the incompatibility is simply to set the **`java.rmi.server.useCodebaseOnly`** property back to **`false`**. This can be done using the following command line options:

```
java -Djava.rmi.server.useCodebaseOnly=false
```

The analogous syntax is used with the **`rmiregistry`** and **`rmid`** commands.

Caution: Running a system with the `java.rmi.server.useCodebaseOnly` property set to `false` is not recommended, as it may allow the loading and execution of untrusted code.

For More Information

- RMI Documentation for information on RMI.
- Dynamic Code Downloading using Java RMI for information on RMI code loading and the codebase mechanism.
- Permissions in Java SE 7 for information on setting permissions in security policy files.

Copyright © 1993, 2018, Oracle and/or its affiliates. All rights reserved.