## NAME

jstatd − monitor the creation and termination of instrumented Java HotSpot VMs

## SYNOPSIS

**Note:** This command is experimental and unsupported.

**`jstatd`** [*options*]

*options* This represents the **`jstatd`** command−line options. See **Options for the jstatd Command**.

## DESCRIPTION

The **`jstatd`** command is an RMI server application that monitors for the creation and termination of instrumented Java HotSpot VMs and provides an interface to enable remote monitoring tools, **`jstat`** and **`jps`**, to attach to JVMs that are running on the local host and collect information about the JVM process.

The **`jstatd`** server requires an RMI registry on the local host. The **`jstatd`** server attempts to attach to the RMI registry on the default port, or on the port you specify with the **`−p port`** option. If an RMI registry is not found, then one is created within the **`jstatd`** application that's bound to the port that's indicated by the **`−p port`** option or to the default RMI registry port when the **`−p port`** option is omitted. You can stop the creation of an internal RMI registry by specifying the **`−nr`** option.

## OPTIONS FOR THE JSTATD COMMAND

**`−nr`** This option does not attempt to create an internal RMI registry within the **`jstatd`** process when an existing RMI registry isn't found.

**`−p`** *port*
This option sets the port number where the RMI registry is expected to be found, or when not found, created if the **`−nr`** option isn't specified.

**`−r`** *rmiport*
This option sets the port number to which the RMI connector is bound. If not specified a random available port is used.

**`−n`** *rminame*
This option sets the name to which the remote RMI object is bound in the RMI registry. The default name is **`JStatRemoteHost`**. If multiple**`jstatd`** serv ers are started on the same host, then the name of the exported RMI object for each server can be made unique by specifying this option. However, doing so requires that the unique server name be included in the monitoring client's **`hostid`** and **`vmid`** strings.

**`−J`***option*
This option passes a Java **`option`** to the JVM, where the option is one of those described on the reference page for the Java application launcher. For example, **`−J−Xms48m`** sets the startup memory to 48 MB. See **java**.

## SECURITY

The **`jstatd`** server can monitor only JVMs for which it has the appropriate native access permissions. Therefore, the **`jstatd`** process must be running with the same user credentials as the target JVMs. Some user credentials, such as the root user in Linux and OS X operating systems, have permission to access the instrumentation exported by any JVM on the system. A **`jstatd`** process running with such credentials can monitor any JVM on the system, but introduces additional security concerns.

The **`jstatd`** server doesn't provide any authentication of remote clients. Therefore, running a **`jstatd`** server process exposes the instrumentation export by all JVMs for which the **`jstatd`** process has access permissions to any user on the network. This exposure might be undesirable in your environment, and therefore, local security policies should be considered before you start the **`jstatd`** process, particularly in production environments or on networks that aren't secure.

The **`jstatd`** server installs an instance of **`RMISecurityPolicy`** when no other security manager is installed, and therefore, requires a security policy file to be specified. The policy file must conform to Default Policy Implementation and Policy File Syntax.

If your security concerns can't be addressed with a customized policy file, then the safest action is to not run the **jstatd** server and use the **jstat** and **jps** tools locally. However, when using **jps** to get a list of instrumented JVMs, the list will not include any JVMs running in docker containers.

## REMOTE INTERFACE

The interface exported by the **jstatd** process is proprietary and guaranteed to change. Users and developers are discouraged from writing to this interface.

## EXAMPLES

The following are examples of the **jstatd** command. The **jstatd** scripts automatically start the server in the background.

## INTERNAL RMI REGISTRY

This example shows how to start a **jstatd** session with an internal RMI registry. This example assumes that no other server is bound to the default RMI registry port (port **1099**).

```
jstatd −J−Djava.security.policy=all.policy
```

## EXTERNAL RMI REGISTRY

This example starts a **jstatd** session with an external RMI registry.

```
rmiregistry&
jstatd −J−Djava.security.policy=all.policy
```

This example starts a **jstatd** session with an external RMI registry server on port **2020**.

```
jrmiregistry 2020&
jstatd −J−Djava.security.policy=all.policy −p 2020
```

This example starts a **jstatd** session with an external RMI registry server on port **2020** and JMX connector bound to port **2021**.

```
jrmiregistry 2020&
jstatd −J−Djava.security.policy=all.policy −p 2020 −r 2021
```

This example starts a **jstatd** session with an external RMI registry on port 2020 that's bound to **AlternateJstatdServerName**.

```
rmiregistry 2020&
jstatd −J−Djava.security.policy=all.policy −p 2020 −n AlternateJstatdServer
```

## STOP THE CREATION OF AN IN−PROCESS RMI REGISTRY

This example starts a **jstatd** session that doesn't create an RMI registry when one isn't found. This example assumes an RMI registry is already running. If an RMI registry isn't running, then an error message is displayed.

```
jstatd −J−Djava.security.policy=all.policy −nr
```

## ENABLE RMI LOGGING

This example starts a **jstatd** session with RMI logging capabilities enabled. This technique is useful as a troubleshooting aid or for monitoring server activities.

```
jstatd −J−Djava.security.policy=all.policy −J−Djava.rmi.serv-
er.logCalls=true
```