NAME

java − the Java application launcher

SYNOPSIS

java [options] class [argument ...]
java [options] **−jar** file.jar [argument ...]

options
Command−line options.

class
Name of the class to be invoked.

file.jar
Name of the jar file to be invoked. Used only with *−jar*.

argument
Argument passed to the **main** function.

DESCRIPTION

The **java** tool launches a Java application. It does this by starting a Java runtime environment, loading a specified class, and invoking that class's **main** method.

The method must be declared public and static, it must not return any value, and it must accept a *String* array as a parameter. The method declaration must look like the following:

public static void main(String args[])

By default, the first non−option argument is the name of the class to be invoked. A fully−qualified class name should be used. If the **−jar** option is specified, the first non−option argument is the name of a **JAR** archive containing class and resource files for the application, with the startup class indicated by the **Main−Class** manifest header.

The Java runtime searches for the startup class, and other classes used, in three sets of locations: the bootstrap class path, the installed extensions, and the user class path.

Non−option arguments after the class name or JAR file name are passed to the **main** function.

OPTIONS

The launcher has a set of standard options that are supported on the current runtime environment and will be supported in future releases. In addition, the current implementations of the virtual machines support a set of non−standard options that are subject to change in future releases.

Standard Options

−client

Select the Java HotSpot Client VM. A 64−bit capable jdk currently ignores this option and instead uses the Java Hotspot Server VM.

For default VM selection, see Server−Class Machine Detection

−server

Select the Java HotSpot Server VM. On a 64−bit capable jdk only the Java Hotspot Server VM is supported so the −server option is implicit.

For default VM selection, see Server−Class Machine Detection

−agentlib:libname[=options]
Load native agent library *libname*, e.g.

−agentlib:hprof

−agentlib:jdwp=help

−agentlib:hprof=help

For more information, see JVMTI Agent Command Line Options.

−agentpath:pathname[=options]
Load a native agent library by full pathname. For more information, see JVMTI Agent Command Line Options.

−classpath classpath

−cp classpath
Specify a list of directories, JAR archives, and ZIP archives to search for class files. Class path entries are separated by colons (**:**). Specifying **−classpath** or **−cp** overrides any setting of the **CLASSPATH** environment variable.

If **−classpath** and **−cp** are not used and **CLASSPATH** is not set, the user class path consists of the current directory (**.**).

As a special convenience, a class path element containing a basename of * is considered equivalent to specifying a list of all the files in the directory with the extension *.jar* or *.JAR* (a java program cannot tell the difference between the two invocations).
For example, if directory *foo* contains *a.jar* and *b.JAR*, then the class path element *foo/** is expanded to a *A.jar:b.JAR*, except that the order of jar files is unspecified. All jar files in the specified directory, even hidden ones, are included in the list. A classpath entry consisting simply of * expands to a list of all the jar files in the current directory. The *CLASSPATH* environment variable, where defined, will be similarly expanded. Any classpath wildcard expansion occurs before the Java virtual machine is started −− no Java program will ever see unexpanded wildcards except by querying the environment. For example; by invoking *System.getenv("CLASSPATH")*.

For more information on class paths, see Setting the Class Path.

−Dproperty=value
Set a system property value.

−d32

−d64
Request that the program to be run in a 32−bit or 64−bit environment, respectively. If the requested environment is not installed or is not supported, an error is reported.

Currently only the Java HotSpot Server VM supports 64−bit operation, and the "−server" option is implicit with the use of −d64. And the "−client" option is ignored with the use of −d64. This is subject to change in a future release.

If neither **−d32** nor **−d64** is specified, the default is to run in a 32−bit environment, except for 64−bit only systems. This is subject to change in a future release.

−enableassertions[:<package name>"..." | :<class name>]

−ea[:<package name>"..." | :<class name>]
Enable assertions. Assertions are disabled by default.

With no arguments, **enableassertions** or **−ea** enables assertions. With one argument ending in *"..."*, the switch enables assertions in the specified package and any subpackages. If the argument is simply *"..."*, the switch enables assertions in the unnamed package in the current working directory. With one argument not ending in *"..."*, the switch enables assertions in the specified class.

If a single command line contains multiple instances of these switches, they are processed in order before loading any classes. So, for example, to run a program with assertions enabled only in package *com.wombat.fruitbat* (and any subpackages), the following command could be used:

java −ea:com.wombat.fruitbat... <Main Class>

The **−enableassertions** and **−ea** switches apply to *all* class loaders and to system classes (which do not have a class loader). There is one exception to this rule: in their no−argument form, the switches do *not* apply to system. This makes it easy to turn on asserts in all classes except for system classes. A separate switch is provided to enable asserts in all system classes; see **−enablesystemassertions** below.

−disableassertions[:<package name>"..." | :<class name>]

−da[:<package name>"..." | :<class name>]
Disable assertions. This is the default.

With no arguments, **disableassertions** or **−da** disables assertions. With one argument ending in *"..."*, the switch disables assertions in the specified package and any subpackages. If the argument is simply *"..."*, the switch disables assertions in the unnamed package in the current working directory. With one argument not ending in *"..."*, the switch disables assertions in the specified class.

To run a program with assertions enabled in package *com.wombat.fruitbat* but disabled in class *com.wombat.fruitbat.Brickbat*, the following command could be used:

java −ea:com.wombat.fruitbat... −da:com.wombat.fruitbat.Brickbat <Main Class>

The **−disableassertions** and **−da** switches apply to *all* class loaders and to system classes (which do not have a class loader). There is one exception to this rule: in their no−argument form, the switches do *not* apply to system. This makes it easy to turn on asserts in all classes except for system classes. A separate switch is provided to enable asserts in all system classes; see **−disablesystemassertions** below.

−enablesystemassertions

−esa
Enable asserts in all system classes (sets the *default assertion status* for system classes to *true*).

−disablesystemassertions

−dsa
Disables asserts in all system classes.

−jar
Execute a program encapsulated in a JAR file. The first argument is the name of a JAR file instead of a startup class name. In order for this option to work, the manifest of the JAR file must contain a line of the form **Main−Class: *classname***. Here, *classname* identifies the class having the *public static void main(String[] args)* method that serves as your application's starting point. See the Jar tool reference page and the Jar trail of the *Java Tutorial* @ http://java.sun.com/docs/books/tutorial/jar for information about working with Jar files and Jar−file manifests.

When you use this option, the JAR file is the source of all user classes, and other user class path settings are ignored.

Note that JAR files that can be run with the "java −jar" option can have their execute permissions set so they can be run without using "java −jar". Refer to Java Archive (JAR) Files.

−javaagent:jarpath[=options]
Load a Java programming language agent, see java.lang.instrument.

−verbose

−verbose:class
Display information about each class loaded.

−verbose:gc
Report on each garbage collection event.

−verbose:jni
Report information about use of native methods and other Java Native Interface activity.

−version
Display version information and exit.

−version:release
Specifies that the version specified by *release* is required by the class or jar file specified on the command line. If the version of the java command invoked does not meet this specification and an appropriate implementation is found on the system, the appropriate implementation will be used.

release not only can specify an exact version, but can also specify a list of versions called a version string. A version string is an ordered list of version ranges separated by spaces. A version range is either a version−id, a version−id followed by a star (*), a version−id followed by a plus sign (+) , or two version−ranges combined using an ampersand (&). The star means prefix match, the plus sign means this version or greater, and the ampersand means the logical anding of the two version−ranges. For example:

−version:"1.5.0_04 1.5*&1.5.1_02+"

The meaning of the above is that the class or jar file requires either version 1.5.0_02, or a version with 1.5 as a version−id prefix and that is not less than 1.5.1_02. The exact syntax and definition of version strings may be found in Appendix A of the Java Network Launching Protocol & API Specification (JSR−56).

For jar files, the usual preference is to specify version requirements in the jar file manifest rather than on the command line.

See the following NOTES section for important policy information on the use of this option.

−showversion
Display version information and continue.

−?

−help
Display usage information and exit.

−X
Display information about non−standard options and exit.

Non−Standard Options

−Xint
Operate in interpreted−only mode. Compilation to native code is disabled, and all bytecodes are executed by the interpreter. The performance benefits offered by the Java HotSpot VMs' adaptive compiler will not be present in this mode.

−Xbatch
Disable background compilation. Normally the VM will compile the method as a background task, running the method in interpreter mode until the background compilation is finished. The *−Xbatch* flag disables background compilation so that compilation of all methods proceeds as a foreground task until completed.

−Xbootclasspath:bootclasspath
Specify a colon−separated list of directories, JAR archives, and ZIP archives to search for boot class files. These are used in place of the boot class files included in the Java 2 SDK. *Note: Applications that use this option for the purpose of overriding a class in rt.jar should not be deployed as doing so would contravene the Java 2 Runtime Environment binary code license.*

−Xbootclasspath/a:path
Specify a colon−separated path of directires, JAR archives, and ZIP archives to append to the default bootstrap class path.

−Xbootclasspath/p:path
Specify a colon−separated path of directires, JAR archives, and ZIP archives to prepend in front of the default bootstrap class path. *Note: Applications that use this option for the purpose of overriding a class in rt.jar should not be deployed as doing so would contravene the Java 2 Runtime Environment binary code license.*

−Xcheck:jni
Perform additional checks for Java Native Interface (JNI) functions. Specifically, the Java Virtual Machine validates the parameters passed to the JNI function as well as the runtime environment data before processing the JNI request. Any invalid data encountered indicates a problem in the native code, and the Java Virtual Machine will terminate with a fatal error in such cases. Expect a performance degradation when this option is used.

−Xfuture
Perform strict class−file format checks. For purposes of backwards compatibility, the default format checks performed by the Java 2 SDK's virtual machine are no stricter than the checks performed by 1.1.x versions of the JDK software. The **−Xfuture** flag turns on stricter class−file format checks that enforce closer conformance to the class−file format specification. Developers are encouraged to use this flag when developing new code because the stricter checks will become the default in future releases of the Java application launcher.

−Xnoclassgc
Disable class garbage collection. Use of this option will prevent memory recovery from loaded classes thus increasing overall memory usage. This could cause OutOfMemoryError to be thrown in some applications.

−Xincgc
Enable the incremental garbage collector. The incremental garbage collector, which is off by default, will reduce the occasional long garbage−collection pauses during program execution. The incremental garbage collector will at times execute concurrently with the program and during such times will reduce the processor capacity available to the program.

−Xloggc:file
Report on each garbage collection event, as with −verbose:gc, but log this data to *file*. In addition to the information *−verbose:gc* gives, each reported event will be preceeded by the time (in seconds) since the first garbage−collection event.

Always use a local file system for storage of this file to avoid stalling the JVM due to network latency. The file may be truncated in the case of a full file system and logging will continue on the truncated file. This option overrides *−verbose:gc* if both are given on the command line.

−Xmsn
Specify the initial size, in bytes, of the memory allocation pool. This value must be a multiple of 1024 greater than 1MB. Append the letter *k* or *K* to indicate kilobytes, or *m* or *M* to indicate megabytes. The default value is chosen at runtime based on system configuration. For more information, see HotSpot Ergonomics
Examples:

−Xms6291456
−Xms6144k
−Xms6m

−Xmxn
Specify the maximum size, in bytes, of the memory allocation pool. This value must a multiple of 1024 greater than 2MB. Append the letter *k* or *K* to indicate kilobytes, or *m* or *M* to indicate megabytes. The default value is chosen at runtime based on system configuration. For more information, see HotSpot Ergonomics
Examples:

−Xmx83886080
−Xmx81920k
−Xmx80m

On Solaris 7 and Solaris 8 SPARC platforms, the upper limit for this value is approximately 4000m minus overhead amounts. On Solaris 2.6 and x86 platforms, the upper limit is approximately 2000m minus overhead amounts. On Linux platforms, the upper limit is approximately 2000m minus overhead amounts.

−Xprof
Profiles the running program, and sends profiling data to standard output. This option is provided as a utility that is useful in program development and is not intended to be be used in production systems.

−Xrs
Reduces use of operating−system signals by the Java virtual machine (JVM).

In a previous release, the Shutdown Hooks facility was added to allow orderly shutdown of a Java application. The intent was to allow user cleanup code (such as closing database connections) to run at shutdown, even if the JVM terminates abruptly.

Sun's JVM catches signals to implement shutdown hooks for abnormal JVM termination. The JVM uses SIGHUP, SIGINT, and SIGTERM to initiate the running of shutdown hooks.

The JVM uses a similar mechanism to implement the pre−1.2 feature of dumping thread stacks for debugging purposes. Sun's JVM uses SIGQUIT to perform thread dumps.

Applications embedding the JVM frequently need to trap signals like SIGINT or SIGTERM, which can lead to interference with the JVM's own signal handlers. The **−Xrs** command−line option is available to address this issue. When **−Xrs** is used on Sun's JVM, the signal masks for SIGINT, SIGTERM, SIGHUP, and SIGQUIT are not changed by the JVM, and signal handlers for these signals are not installed.

There are two consequences of specifying **−Xrs**:

- SIGQUIT thread dumps are not available.
- User code is responsible for causing shutdown hooks to run, for example by calling System.exit() when the JVM is to be terminated.

−Xssn
Set thread stack size.

−XX:+UseAltSigs
The VM uses *SIGUSR1* and *SIGUSR2* by default, which can sometimes conflict with applications that signal−chain *SIGUSR1* and *SIGUSR2*. The *−XX:+UseAltSigs* option will cause the

VM to use signals other than *SIGUSR1* and *SIGUSR2* as the default.

NOTES

The **−version:***release* command line option places no restrictions on the complexity of the release specification. However, only a restricted subset of the possible release specifications represent sound policy and only these are fully supported. These policies are:

1. Any version, represented by not using this option.
2. Any version greater than an arbitrarily precise version−id. For example:

   "1.5.0_03+"

   Would utilize any version greater than 1.5.0_03. This is useful for a case where an interface was introduced (or a bug fixed) in the release specified.
3. A version greater than an arbitrarily precise version−id, bounded by the upper bound of that release family. For example:

   "1.5.0_03+&1.5*"

4. "Or" expressions of items 2. or 3. above. For example:

   "1.4.2_05+&1.4* 1.5+"

   Similar to item 2. this is useful when a change was introduced in a release (1.5) but also made available in updates to previous releases.

SEE ALSO

- javac − the Java programming language compiler
- jdb − Java Application Debugger
- javah − C Header and Stub File Generator
- jar − JAR Archive Tool
- The Java Extensions Framework
- Security Features.
- *HotSpot VM Specific Options @* http://java.sun.com/docs/hotspot/VMOptions.html.