NAME

java − Java interpreter

SYNOPSIS

java [*options*] *class* [*argument ...*]

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

PARAMETERS

Options may be in any order. For a discussion of parameters which apply to a specific option, see **OPTIONS** below.

options	Command-line options. See **OPTIONS** below.
class	Name of the class to be invoked.
file.jar	Name of the jar file to be invoked. Used only with the **−jar** option.

DESCRIPTION

The **java** utility 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 have the following signature:

public static void main(String args[])

The method must be declared **public** and **static**, it must not return any value, and it must accept a **String** array as a parameter. 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. However, options below that are described as having been replaced by another one are obsolete and may be removed in a future release. An additional set of non-standard options are specific to the current virtual machine implementation and are subject to change in the future. Non-standard options begin with **−X**.

Standard Options

−client
Selects the Java HotSpot Client VM. This is the default.

−server
Selects the Java HotSpot Server VM.

−classpath *classpath*
−cp *classpath*
Specifies 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.

Used with **java**, the **−classpath** or **−cp** options only specify the class path for user classes. Used with **−classpath** or **−cp** specify the class path for both user classes and bootstrap classes.

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

−debug
This has been replaced by **−Xdebug**.

−D*property=value*
Sets a system property value.

−enableassertions :<package name>... |:<class name>

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

With no arguments, **enableassertions** or **−ea** enable 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 **s** 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;
−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 rent 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 **ss** 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 Executes a program encapsulated in a JAR archive. 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 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.

−noclassgc This has been replaced by **−Xnoclassgc**.

−ms*n* This has been replaced by **−Xms** *n*.

−mx*n* This has been replaced by **−Xmx** *n*.

−ss*n* This has been replaced by **−Xss** *n*.

−verbose

−verbose:*class* Displays information about each class loaded.

−verbosegc This has been replaced by **−verbose:gc**.

−verbose:gc Reports on each garbage collection event.

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

−version Displays version information and exit.

−showversion Displays version information and continues.

−?

−help Displays usage information and exit.

−X Displays information about non-standard options and exit.

Non-Standard Options

−Xint Operates 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.

−Xbootclasspath:*bootclasspath*
Specifies 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 and Java 2 Runtime Environment.

−Xbootclasspath/a:*path*
Specifies a colon-separated *path* of directories, JAR archives, and ZIP archives to append to the default bootstrap class path.

−Xbootclasspath/p:*path*
Specifies a colon-separated *path* of directories, 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.

−Xdebug Starts with the debugger enabled.

−Xcheck:jni Perform additional check for Java Native Interface functions.

−Xfuture Performs 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 Disables class garbage collection

−Xincgc
: Enable the incremental garbage collector. The incremental garbage collector, which is off by default, will eliminate occasional garbage-collection pauses during program execution. However, it can lead to a roughly 10% decrease in overall GC performance.

−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.

−Xms*n*
: Specifies the initial size of the memory allocation pool. This value must be greater than 1000. To modify the meaning of *n*, append either the letter **k** for kilobytes or the letter **m** for megabytes. The default value is 2m.

−Xmx*n*
: Specifies the maximum size of the memory allocation pool. This value must be greater than 1000. To modify the meaning of *n*, append either the letter **k** for kilobytes or the letter **m** for megabytes. The default value is 64m. The uppoer limit for this value will be approximately 4000m on SPARC platforms and 2000m on x86 platforms, 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.

−Xrunhprof[:help][:*suboption*=*value*,...]
: Enables cpu, heap, or monitor profiling. This option is typically followed by a list of comma-separated *suboption*=*value* pairs. Run the command **java −Xrunhprof:help** to obtain a list of suboptions and their default values.

−Xss*n*
: Each Java thread has two stacks: one for Java code and one for C code. The **−Xss** option sets the maximum stack size that can be used by C code in a thread to *n*. Every thread that is spawned during the execution of the program passed to **java** has *n* as its C stack size. The default units for *n* are bytes and *n* must be > 1000 bytes.

  To modify the meaning of *n*, append either the letter **k** for kilobytes or the letter **m** for megabytes. The default stack size is determined by the Linux operating system upon which the Java platform is running.

−Xrs
: Reduce usage of operating-system signals by Java virtual machine (JVM).

  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 SIGQUIT to perform thread dumps.

  Applications that embed the JVM frequently need to trap signals like SIGINT or SIGTERM, and in such cases there is the possibility of interference between the applications' signal handlers and the JVM shutdown-hooks facility.

  To avoid such interference, the **−Xrs** option can be used to turn off the JVM shutdown-hooks feature. When **−Xrs** is used, the signal masks for SIGINT, SIGTERM, SIGHUP, and SIGQUIT are not changed by the JVM, and signal handlers for these signals are not installed.

ENVIRONMENT VARIABLES

CLASSPATH
: Used to provide the system with a path to user-defined classes. Directories are separated by colons. For example:

.:/home/avh/classes:/usr/local/java/classes

SEE ALSO

javac(1), **jdb**(1), **javac**(1), **jar**(1), **set**(1)

See (or search **java.sun.com**) for the following:

JDK File Structure @
http://java.sun.com/j2se/1.4/docs/tooldocs/linux/jdkfiles.html

JAR Files @
http://java.sun.com/docs/books/tutorial/jar/

NOTES

All the **−X** options are unstable. As noted in the **OPTIONS** section, some of the "standard" options are obsolete.