NAME

jdb − find and fix bugs in Java platform programs

SYNOPSIS

`jdb` [*options*] [*classname*] [*arguments*]

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

classname
This represents the name of the main class to debug.

arguments
This represents the arguments that are passed to the **`main()`** method of the class.

DESCRIPTION

The Java Debugger (JDB) is a simple command−line debugger for Java classes. The **`jdb`** command and its options call the JDB. The **`jdb`** command demonstrates the Java Platform Debugger Architecture and provides inspection and debugging of a local or remote JVM.

START A JDB SESSION

There are many ways to start a JDB session. The most frequently used way is to have the JDB launch a new JVM with the main class of the application to be debugged. Do this by substituting the**`jdb`** command for the **`java`** command in the command line. For example, if your application's main class is **`MyClass`**, then use the following command to debug it under the JDB:

```
jdb MyClass
```

When started this way, the **`jdb`** command calls a second JVM with the specified parameters, loads the specified class, and stops the JVM before executing that class's first instruction.

Another way to use the **`jdb`** command is by attaching it to a JVM that's already running. Syntax for starting a JVM to which the **`jdb`** command attaches when the JVM is running is as follows. This loads in−process debugging libraries and specifies the kind of connection to be made.

```
java −agentlib:jdwp=transport=dt_socket,server=y,suspend=n MyClass
```

You can then attach the **`jdb`** command to the JVM with the following command:

```
jdb −attach 8000
```

8000 is the address of the running JVM.

The **`MyClass`** argument isn't specified in the **`jdb`** command line in this case because the **`jdb`** command is connecting to an existing JVM instead of launching a new JVM.

There are many other ways to connect the debugger to a JVM, and all of them are supported by the **`jdb`** command. The Java Platform Debugger Architecture has additional documentation on these connection options.

BREAKPOINTS

Breakpoints can be set in the JDB at line numbers or at the first instruction of a method, for example:

- The command **`stop at MyClass:22`** sets a breakpoint at the first instruction for line 22 of the source file containing **`MyClass`**.
- The command **`stop in java.lang.String.length`** sets a breakpoint at the beginning of the method **`java.lang.String.length`**.
- The command **`stop in MyClass.<clinit>`** uses **`<clinit>`** to identify the static initialization code for **`MyClass`**.

When a method is overloaded, you must also specify its argument types so that the proper method can be selected for a breakpoint. For example, **`MyClass.myMethod(int,java.lang.String)`** or **`MyClass.myMethod()`**.

The **`clear`** command removes breakpoints using the following syntax: **`clear MyClass:45`**. Using the **`clear`** or **`stop`** command with no argument displays a list of all breakpoints currently set. The **`cont`**

command continues execution.

STEPPING

The **step** command advances execution to the next line whether it's in the current stack frame or a called method. The**next** command adv ances execution to the next line in the current stack frame.

EXCEPTIONS

When an exception occurs for which there isn't a **catch** statement anywhere in the throwing thread's call stack, the JVM typically prints an exception trace and exits. When running under the JDB, however, control returns to the JDB at the offending throw. You can then use the **jdb** command to diagnose the cause of the exception.

Use the **catch** command to cause the debugged application to stop at other thrown exceptions, for example: **catch java.io.FileNotFoundException** or **catch mypackage.BigTroubleException**. Any exception that's an instance of the specified class or subclass stops the application at the point where the exception is thrown.

The **ignore** command negates the effect of an earlier **catch** command. The **ignore** command doesn't cause the debugged JVM to ignore specific exceptions, but only to ignore the debugger.

OPTIONS FOR THE JDB COMMAND

When you use the **jdb** command instead of the **java** command on the command line, the **jdb** command accepts many of the same options as the **java** command.

The following options are accepted by the **jdb** command:

-help Displays a help message.

-sourcepath *dir1***:***dir2***:***...*
Uses the specified path to search for source files in the specified path. If this option is not specified, then use the default path of dot (**.**).

-attach *address*
Attaches the debugger to a running JVM with the default connection mechanism.

-listen *address*
Waits for a running JVM to connect to the specified address with a standard connector.

-listenany
Waits for a running JVM to connect at any available address using a standard connector.

-launch
Starts the debugged application immediately upon startup of the **jdb** command. The **-launch** option removes the need for the **run** command. The debugged application is launched and then stopped just before the initial application class is loaded. At that point, you can set any necessary breakpoints and use the **cont** command to continue execution.

-listconnectors
Lists the connectors available in this JVM.

-connect *connector-name***:***name1***=***value1*...
Connects to the target JVM with the named connector and listed argument values.

-dbgtrace [*flags*]
Prints information for debugging the **jdb** command.

-tclient
Runs the application in the Java HotSpot VM client.

-tserver
Runs the application in the Java HotSpot VM server.

-J*option*
Passes *option* to the JVM, where option is one of the options described on the reference page for the Java application launcher. For example, **-J-Xms48m** sets the startup memory to 48 MB. See

Overview of Java Options in **java**.

The following options are forwarded to the debuggee process:

-v or **-verbose**[**:***class*|**gc**|**jni**]
Turns on the verbose mode.

-D*name***=***value*
Sets a system property.

-classpath *dir*
Lists directories separated by colons in which to look for classes.

-X *option*
A nonstandard target JVM option.

Other options are supported to provide alternate mechanisms for connecting the debugger to the JVM that it's to debug.