Partner Resources

With JDWP

Take a look at this simple breakdown of how to use the JDWP to debug Java applications.

By Mahmoud Anouti MVB · Jul. 16, 19 · Tutorial

Most Java developers have had to debug their applications, usually to find and fix an issue there. In many cases, the application to debug (known as the *"debuggee"*) is launched from within the IDE used by the developer, while the debugger is also integrated into the IDE, allowing easy inspection of the program state in a step-by-step manner. Sometimes, however, the debuggee JVM is launched from a separate command line, or by executing it on a separate host. In such scenarios, debugging necessitates launching the JVM with some options suitable for debugging, while your IDE debugger would have to connect to it. This is where JDWP (Java Debug Wire Protocol) comes into play.

What is JDWP?

In order to debug remotely executed JVMs (where the debuggee is separately launched locally or on another machine), the Java platform defines a protocol for communication between the JVM and the debugger. JDWP dictates the format of the commands sent by the debugger (e.g. to evaluate a local variable), and replies by the JVM. The exact way of transporting the packets is not specified and is up to the implementation to define transport mechanisms. What JDWP specifies is the format and layout of packets containing commands and those containing replies. Therefore it is conceptually very simple.

JDWP is only one part of the debugging infrastructure in the Java platform. The endpoints (debugger and debuggee) communicating over JDWP implement other specifications to provide

d
hi
in
A

Ir
tr
(
co
sh s
d
o
so

Debugging Remotely Using JDWP

The way a debugger connects to a target JVM is by having one act as the server listening for an incoming connection, and the other attaching to the server. The server endpoint could be either the JVM or the debugger. This applies to both socket transport and shared memory transport. Therefore, there are two steps to perform in order to remotely debug a target app:

1. Launch either the JVM or the debugger in server mode, so that it listens at a certain address, namely an assigned IP address and port number.
2. Attach the other part to the listening server on that address.

For example, to launch the JVM with debug options to listen on an address, we use the following option with the Java executable:

```
java -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=8000  ... MainClass
```

The `-agentlib:jdwp` with the comma-separated key-value suboptions instruct the JVM to load

- `server=y` means that the JVM will listen for a debugger to attach to it.
- `suspend=y` means the JVM will wait for the debugger to attach before executing the main class. This is also the default value. If set to *n*, the JVM will immediately execute the main class, while listening for the debugger connection.
- `address=8000` specifies the address at which the debug socket will listen. In this case, the JVM will listen at port 8000 for incoming connections **only from the local host (starting JDK 9)**.

The second step is to attach the debugger at that address. All popular IDEs provide a way to easily do this. In Eclipse, for example, it can be configured by going to Run -> Debug Configuration and creating a Remote Java Application configuration:

2019-07-07_17_22_26-Eclipse_remote_debug_attach

Notice that the host and port must match the address of the JDWP agent on JVM side.

[JDK 9+] Binding the Listening Socket to All Addresses

In the previous example, the `address` was set to 8000 (port number) without any hostname or IP address. Before JDK 9, this would mean the JVM would listen on all available IP addresses, making the socket accessible by debuggers on remote machines. Starting with JDK 9, this was changed to only allow local connections for better security. In other words, `-agentlib:jdwp=transport=dt_socket,server=y,address=8000` is now equivalent to `-agentlib:jdwp=transport=dt_socket,server=y,address=localhost:8000`.

To bind the socket to addresses allowing remote connections, either prefix the port with the host name, IP address, or an asterisk (`*`) to bind to all available IP addresses:

```
-agentlib:jdwp=transport=dt_socket,server=y,address=host1:8000
```

or

Adding a Timeout

We can add a timeout for the JDWP agent listening for the debugger. To make the JVM exit after 10 seconds without any debugger attaching:

```
-agentlib:jdwp=transport=dt_socket,server=y,address=*:8000,timeout=10000
```

Listening at A Dynamic Port

If `server=y` (i.e. JVM is listening for connection), we can skip the `address` option, which will make it use a dynamically assigned port. Since no address was specified, this allows only local connections. The chosen port will be displayed at stdout of the JVM, e.g.:

```
Listening for transport dt_socket at address: 12345
```

The Other Way Around: Attaching to A Debugger

We can set `server=n` on the JVM command line option (or just remove the `server` option as it defaults to `n`), and tell it to attach to a debugger at a certain address. We would first run the debugger in listening mode:

2019-07-07_17_22_26-Eclipse_remote_debug_listen

Let's say the debugger was started on `host2`. We would then run the JVM with the option:

```
-agentlib:jdwp=transport=dt_socket,address=host2:8000
```

Delaying JDWP Connection Establishment until A Specific Exception Is Thrown

A useful option to the JDWP agent is to start the JVM as normal and wait until a specific exception is thrown. For example, say you want to debug a failing application with a `MyCustomException` but don't want to initiate the debugger connection until it is thrown. This

```
agentlib:jdwp=transport=dt_socket,server=y,address=*:8000,onthrow=com.example.MyCustom
Exception,launch=notify_script
```

This would start the application normally without listening on the address. When the exception is thrown, the agent will listen on port 8000 and a debugger can be attached to it. The `launch` option is a mandatory option along with `onthrow` used to start a certain process when the exception is thrown. The process will be given the transport and port number as arguments. It can be used, for example, to automatically launch the debugger to attach to the listening VM upon the exception being thrown.

References

- Java Platform Debugger Architecture
- JDWP spec
- JPDA Connection and Invocation Details
- [JDK 9 Release Notes] JDWP socket connector accept only local connections by default

Java (Programming Language) Application Java Virtual Machine Remote Shared Memory Connection (Dance) Debug (Command)

RELATED

JVM Memory Architecture and GC Algorithm Basics

How to Configure, Customize, and Use Ballerina Logs