TRENDING

RAG Systems: A Brand New Architecture Tool

Lifecycle Microservices With GenAI Tools

Leveraging Event-Driven Data Mesh Architecture With AWS for Modern Data nges

's Article Submission Guidelines

Remote Debugging Java Applications 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 cas
"debuggee") is launched from
debugger is also integrated int
program state in a step-by-ste
JVM is launched from a separa
host. In such scenarios, debugg
options suitable for debugging
to it. This is where JDWP (Java

What is JDWP?

In order to debug remotely ex
launched locally or on another
for communication between th
format of the commands sent
and replies by the JVM. The ex
specified and is up to the impl
What JDWP specifies is the for
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 the actual debugging functionality. The JVM implements the JVM Tool Interface (JVMTI) to provide debugging functionality for it, for example, to control executions using breakpoints or inspecting the current object. JVMTI is the low-level layer implemented natively in the JVM. The debugger implements another interface called the Java Debug Interface (JDI) that provides a high-level way to carry debugging requests from the debugger process. JDI is a pure Java interface. Together, JVMTI, JDWP, and JDI form the main layers of the Java Platform Debugger Architecture. Links to official references about all these specifications are provided at the end.

In the Oracle Java implementation, there are two transport mechanisms provided: the socket transport, and the shared memory transport for Windows only. The socket transport (`dt_socket`) relies on TCP sockets bound to listen on a port for connections, and using that connection to transfer the debug session packets. Shared memory transport (`dt_shmem`) uses shared memory to send

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
the JVM to load the JDWP agen
Here's what each suboption do

- `transport=dt_socket` tel
- `server=y` means that the
- `suspend=y` means the JV
executing the main class.
will immediately execute
connection.
- `address=8000` specifies t
In this case, the JVM will l
from the local host (sta

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

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

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

More Examples

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 Dynan

If `server=y` (i.e. JVM is listenir
option, which will make it use
was specified, this allows only
displayed at stdout of the JVM

```
Listening for transport dt_
```

The Other Way Arou

We can set `server=n` on the JV
`server` option as it defaults to
address. We would first run th

2019-07-07_17

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 can be done with the `onthrow` option:

```
-agentlib:jdwp=transport=dt_socket,server=y,address=*:8000,onthrow=com.example.MyCustomException,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

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