Advanced Debugging Techniques for Java Apps using IntelliJ IDEA and Remote Debugger

20230621

CIKLUM

CIKLUM

Java Community in Ciklum

CIKLUM

Contents

CIKLUM

04 Remote debugging

05 Debug options for different architectures

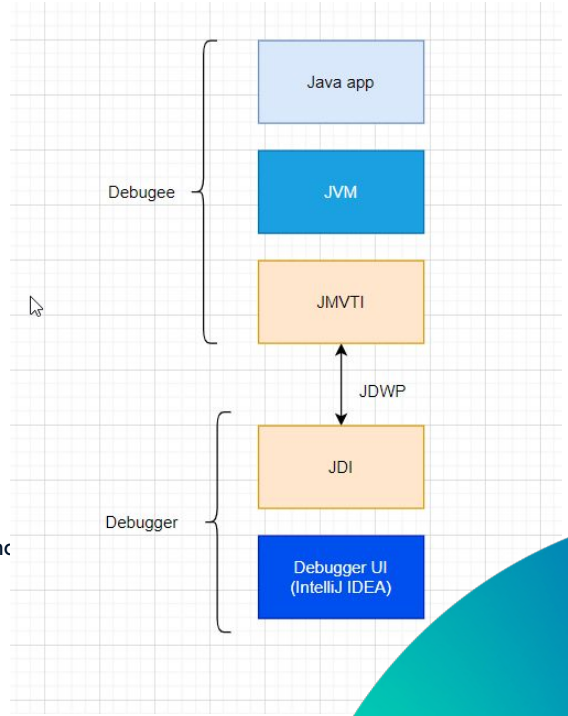
06 Postmortem debugging

07 Q&A

Why debugging?

A Java debugger is a tool for debugging and analyzing Java code during runtime.

- Execute the program step by step and see internal state of the app
- **Find and fix bugs - most common** (but not only!)
- Code analysis
- Change behaviour of the app by changing apps inner state (code or data) on the flight to switch app state
- Patch the app (through HotSwap - **not recommended in prod**)
- Add more logging
- Analyze memory issues
- “Breakpoint bomb”



Java Platform Debugger Architecture (JPDA)

- Java Debugger Interface (JDI) - high-level Java language interface
- Java Virtual Machine Tools Interface (JVMTI) – native interface, used to inspect the state and to control the execution
- Java Debug Wire Protocol (JDWP) – communication Java app and debugger processes

--

JVM References

Improve your apps performance by understanding how JVM works -> slides & recording

JVM & IntelliJ IDEA Debugger - basics

- ❑ Pause
- ❑ Threads/frames/variables
- ❑ Move breakpoint by hand
- ❑ Set multiple same time
- ❑ Run to cursor
- ❑ Different colors for frames
- ❑ Jump to source from variable
- ❑ Evaluate and quick eval
- ❑ Watch
- ❑ Set var val

JVM & IntelliJ IDEA Debugger - Pro tips

- ❑ Breakpoints dialog
  - ❑ Description
  - ❑ Group
  - ❑ Sort
- ❑ Suspend
- ❑ Logging for object init
- ❑ Dependent Breakpoint (when a method is called by another)
- ❑ Breakpoint - all vs thread
- ❑ Breakpoint conditional
- ❑ **Filters**
- ❑ Instance filter
- ❑ Class filter
- ❑ Pass count
- ❑ Intentions
- ❑ Breakpoint types
  - ❑ **Line**
  - ❑ Method
  - ❑ Field
  - ❑ Exception

Remote debugging

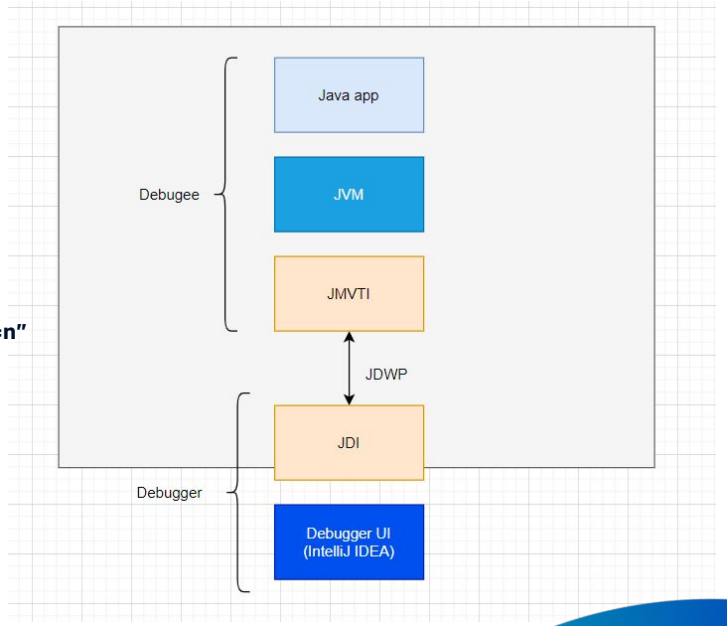
Tomcat

Locate the catalina.sh (for Unix/Linux) or catalina.bat (for Windows) script in the bin directory.

JPDA_OPTS="-agentlib:jdwp=transport=dt_socket,address=<debugger-address>,server=y,suspend=n"

IntelliJ IDEA

// live demo

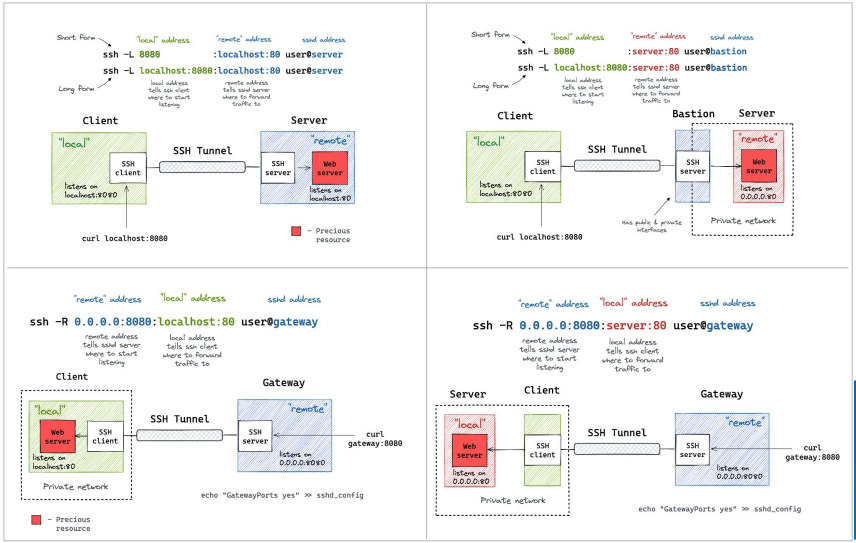


Debug options for different architectures

- An SSH tunnel establishes an encrypted connection between a local machine and a remote server, allowing secure communication between the two.

  How to do it

- Enable remote debugging on host
- **ssh -L <local-port>:localhost:<remote-debug-port> <username>@<remote-server-address>**
- Configure IDE for remote debugging
- Establish SSH tunnel



Source: https://iximiuz.com/en/posts/ssh-tunnels/

Postmortem debug

- **Crash Dumps:**

- Crash dumps are typically generated by Windows operating systems when an application encounters a critical error and crashes.
- To configure crash dumps on **Windows**, you can use the built-in **Windows Error Reporting (WER)** feature or third-party tools.
- WER allows you to configure settings through the Control Panel or Group Policy to specify whether crash dumps should be generated, their location, and the type of crash information to include.

- **Core Dumps:**

- Core dumps are commonly associated with **Unix**-like operating systems such as Linux and macOS.
- To configure core dumps on Linux, you need to adjust the ulimit settings (such as **ulimit -c unlimited**) to allow the creation of core dumps.
- Core dumps are generated when a program encounters a segmentation fault or another fatal error, and they capture the state of the program's memory.

Tools:

- Oracle Troubleshooting guide
- Postmortem diagnostic tools

Learn & Practice References

Code

- ❏ Git repo

Tutorials

- ❏ JVM Specifications
- ❏ SSH tunnel
- ❏ JDB
- ❏ IntelliJ IDEA
- ❏ Visual Studio Code
- ❏ Remote debugging

Thank you!

Lucian Gruia
lugr@ciklum.com

luciangruia.ro