ROOKOUT

Java Application Server Debugging

Java Application Servers are at the heart and soul of many enterprise software stacks. Whether it's a traditional WebLogic or WebSphere server hosting monolith applications or a potentially containerized Tomcat server, Java offers a tried and tested approach for writing enduring software applications. Unfortunately, those servers are not always easy to debug, especially when applications tend to follow on the larger end of the scale.

The Challenges Ahead

Since Java Application Server systems are powerful platforms and can get complex, you need to be aware of three major challenges that you may encounter when attempting to debug them:

- Running the application on a dev server instead of their own laptops, which makes orchestration and debugging harder.
- Slow start times, which makes code changes expensive during debug/development sessions. Customers report start times of over an hour (!) under some conditions.
- Highly multi-threaded environments (often reaching tens of thousands of threads). This can make using a classic debugger more difficult - more on that below.

> ***You'll find that some production debuggers, such as Rookout, go beyond showing the data and offer the ability to export the data to an analytics system of your choice.***

ROOKOUT

Approaches to Java Debugging

There are four main approaches to debugging Java Application Servers- Classic Debuggers, Logging, Logging with Hot Swapping, and Production Debuggers. Each one has its own unique benefits and disadvantages. In this paper, we will explore each one of them by analyzing parameters such as ease of use, stability, impact on performance, and more.

Classic Debuggers

Classic Debuggers, as a debugging approach for Java Application Server systems, have more cons than pros. While Classic Debuggers allow you to skip builds and restarts by setting breakpoints anywhere in the code, their ease of use is often poor, since debugging a process on a remote server tends to be difficult.
Another downside is that Classic Debuggers often have a significant performance overhead, which for large applications can easily become prohibitive. On top of that, breaking into a highly multithreaded application is going to cause various timeouts and synchronization issues which may result in unexpected business logic failures.

Logging

Logging is simple and familiar to most developers from their earliest exposure to computer science, allowing for ease of use. Yet, it isn't a perfect debugging approach.
Once we add the logline to the applications' source-code, we have to rebuild and redeploy to see the change take effect, both of which can take quite a long time.
In terms of application stability and performance, logging has minimal impact, unless an added logline is called too often, throws an exception, or causes an unanticipated side effect. In that case, the problematic logline must be removed, and again the application must be rebuilt and restarted to restore stability.

Logging with Hot Swapping

Logging with hot-swapping is an approach that attempts to bridge one of the main gaps in the traditional logging approach: requiring a server restart. When using hot swapping, you change the source-code, rebuild, and then update the server to use the new version without updating.

While it sounds like magic, it comes with a steep learning curve, and operating hot swappers on a remote server, within a complex Java Application Server, is tricky at best. With great power comes great responsibility, and you can easily introduce new bugs or even crash the server when making so much as a minor mistake operating this tool.

> ***Java servers are not always easy to debug, especially when applications tend to follow on the larger end of the scale.***

ROOKOUT

Production Debugging

Last but most definitely not least of the debugging approaches: production debuggers. They have proven themselves to be quite an effective debugging method. To begin with, their ease of use is unparalleled. All that has to be done is to simply add a Java Agent to your server.
Production debuggers, similar to classic debuggers, allow you to skip rebuilding your code and don't require you to restart your server. They have some of the best performance characteristics around and even include built-in performance protections (for production use).
In terms of application stability, production debuggers do the heavy lifting, enabling devs to insert breakpoints and get the data they need while ensuring their applications remain stable.
Take it one step further and you'll find that some production debuggers, such as Rookout, go beyond showing the data and offer the ability to export the data to an analytics system of your choice, such as AppDynamics or Elasticsearch.

Implementing in Reality

While there are multiple options for debugging Java Application Servers, for most large workloads, you will definitely find that Production Debugging outshines the rest. Superb ease of use combined with production-grade technology makes development environments more accessible than ever.
No more constant need to redeploy and restart, just debugging freely and easily.

Java Server Debugging Approaches

	Ease of Use	Skip Build	Skip Restart	Application Performance	Application Stability	ETL
Classic Debugger	-	+	+	-	+	-
Logging	+	-	-	+	+	-
Logging + Hot Swapping	-	-	+	+	-	-
Production Debugger	+	+	+	+	+	+

TRY FREE