KatoProposal

Abstract

Kato is a project to develop the Specification, Reference Implementation and Technology Compatibility Kit for JSR 326: Post-mortem JVM Diagnostics API

Proposal

JSR 326 is intended to be a Java API specification for standardising how and what can be retrieved from the contents of post-mortem artefacts – typically process and JVM dumps. Project Kato is intended to be the place where the Specification, Reference implementation (RI) and Technology Compatibility Kit (TCK) are openly created. The intention is that the Specification and RI will be developed in tight unison, guided by a user-story-focused approach to ensure that real-world problems drive the project from the beginning.

Unusually for new APIs, this project will endeavour to encompass the old and the new. A diagnostic solution that only works when users move to the latest release does little to improve diagnosability in the short term. This project will consume existing dump artefacts as well as possible while developing an API that can address the emerging trends in JVM and application directions. The most obvious of these trends are the exploitation of very large heaps, alternative languages and, paradoxically for Java, the increased use of native memory through vehicles such as NIO.

Background

Post-mortem versus Live Monitoring: It's worth noting that the term “post mortem” is used loosely. It does not just imply dead JVMs and applications; JSR 326 also covers living, breathing applications where the dump artefacts are deliberately produced as part of live monitoring activity. Live monitoring generally means tracing, profiling, debugging, or even bytecode monitoring and diagnosis by agents via the java.lang.instrument API . It can also mean analysis of dumps to look for trends and gather statistics. The live-monitoring diagnostic space is well served except for this last area, which is where JSR 326 can help.

IBM developed an API called DTFJ (“Diagnostic Tooling and Framework for Java”) as a means of providing its support teams a basis on which to write tools to diagnose Java SDK and Java application faults. It consists of a native JVM-specific component and the DTFJ API, which was written in pure Java.

Rationale

JSR 326 exists because of the widely acknowledged limitations in diagnosing Java application problems after the fact. There are many good ways to understand and diagnose problems while they happen, but few credible or pervasive tools exist for helping resolve problems when all has gone suddenly and horribly wrong. Outside of “live monitoring” there is no standard way to provide diagnostics information, and hence no standard tools. Each tool writer has to figure out how to access the data individually and specifically for each JVM vendor and operating system. This sparsity of tools has meant that users have limited options in diagnosing their own problems, especially unexpected or intermittent failures. Consequently these users turn to the providers of their software to work out what is happening. Application, middleware, and JVM vendors are spending increasing time supporting customers in problem diagnosis. Emerging trends indicate that this is going to get worse.

Today JVM heap sizes are measured in small numbers of gigabytes, processors on desktops come in twos or fours, and most applications running on a JVM are written in Java. To help analyse problems in these configurations, we use a disparate set of diagnostic tools and artefacts. If the problem can't be reproduced in a debugger, then things quickly get complicated. There are point tools for problems like deadlock analysis or the ubiquitous Java out-of-memory problems, but overall the Java diagnostic tools arena is fragmented and JVM- or OS-specific. Tool writers have to choose their place in this matrix. We want to change that by removing the need for tool writers to make a choice. By enabling tool writers to easily target all the major JVM vendors and operating systems, we expect the number and capability of diagnostic tools to greatly increase. Tomorrow it gets harder; heap sizes hit 100's of gigabytes, processors come packaged in powers of 16, and the JVM commonly executes a wide range of language environments. We can't tackle tomorrow's problems until we have a platform to address today's.

This project is about bringing together people and ideas to create that platform, and we can't think of a better place to do that than in Apache.

Initial Goals

The code donated by IBM will not be a complete solution as the JVM-specific portion will be missing. The initial goals will be to close this gap by creating equivalent open-source replacements. We plan to create code to be able to understand a dump related to the Apache Harmony Dynamic Runtime Layer Virtual Machine (DRLVM). We also intend to create an HPROF dump format reader.

After that the expectation is that we work to produce ways to better understand the contents of a process dump. This will allow the work on solving native out-of-memory problems to proceed.

Current Status

Meritocracy

We appreciate how important it is to create and maintain a healthy environment based on meritocracy. We will do all we can to ensure that all opportunities to contribute are visible and that contributions are rewarded appropriately.

Community

In developing a standard it's tempting to get into an small group, write a specification, and then present it to the world. We do not want to do that. We strongly believe that any software interface standard should be developed in as visible a way as possible and show that it has taken the views of the industry into account. We see that the best way to do that for JSR 326 is to develop the specification and implementation in unison, in the open. The JSR expert group is intended to represent the industry, while the RI and example tools will demonstrate the specification's usefulness and applicability. The specification will be developed by incrementally proposing problem statements, constructing specification changes, implementing them within the RI, and demonstrating practicability with example tools. The JSR specification mailing list will be open to everyone to see and contribute to.

The technology that is outlined by this proposal serves to bring standardisation to a fragmented area. We hope and expect that developing this technology will attract the right folks and that a community of data providers and data consumers will develop around it. We are already building relationships with existing tool writers, with the intention of understanding what they need and showing, by their adoption of this API, that we've got it right.

Known Risks

Orphaned Products

The development of, and continuing support for, this project is important to a number of companies. There is no risk of this code being suddenly abandoned.

Inexperience with Open Source

There is a mixed level of experience with Open Source.

Homogeneous Developers

The initial list of committers consists of developers from IBM and one independent. The committers are based in Europe. There is a general level of experience in working in a distributed way. There will be more more developers from other companies joining but they have not yet officially confirmed their participation.

Reliance on Salaried Developers

Most of the developers are paid by their employer to contribute to this project. However the subject matter of this project holds a fascination for many people and over time we expect to see more volunteers contributing.

Relationships with Other Apache Products

Currently the initial seed code uses the Ant build tool only. The need for a suitable open-source JVM from which to harvest dumps, and to explore ways to understand them, requires that this project forge good links with the Apache Harmony project. Since this project is about standardising content and access methods, opportunities also exist to explore closer links with search technologies such as Lucene. There is also potential for the semantics of other Apache projects that use Java, such as Struts or Felix, to be exposed using Kato. For example, a module could be written that describes loaded classes in terms of OSGi bundles instead of just class loaders. It is also possible that the Apache Harmony project may want an implementation of Kato.

Generally we see this project as helping other Apache projects simply because we are improving the diagnostic capabilities of Java. We will welcome all types of diagnostic problem statement and dumps from other Apache projects.

An Excessive Fascination with the Apache Brand

Kato is about creating an API and a community of data providers and data consumers. Creating this API requires that both groups work together as openly as possible.

In fact, Kato lives or dies by the strength of its community. We believe that there is mutual benefit to the project and to the Apache brand if this work is carried out as an Apache project. The project gains by being seen to be serious in its objectives of maintaining a fair, inclusive and equitable approach. Apache benefits from the acceptance of a project that we hope will push technical boundaries and grow to be a community of reasonable size.

Documentation

Information about JSR 326 is available at

http://jcp.org/en/jsr/detail?id=326

Information on IBM's DTFJ can be found at

http://publib.boulder.ibm.com/infocenter/javasdk/v6r0/index.jsp?topic=/com.ibm.java.doc.diagnostics.60/html/dtfj.html

Initial Source

IBM will donate its Diagnostic Tool Framework for Java (DTFJ) technology as a seed for development. This codebase consists of diagnostic dump readers and the data access API. There are data providers for DTFJ that are specific and proprietary to IBM JVMs; these providers will not be open-sourced. DTFJ is being donated to kick start the project and as a foundation on which to build and develop our ideas. There is no expectation that the final output must resemble this seed offering.

Source and Intellectual Property Submission Plan

Apache would receive all source and documentation under the Apache Corporate Contributor agreement. IBM is the only license holder.

External Dependencies

Ant and JUnit are the only external dependencies.

Cryptography

Required Resources

Mailing Lists

- kato-private
- kato-dev
- kato-spec

Subversion Directory

https://svn.apache.org/repos/asf/incubator/kato

Issue Tracking

JIRA : Kato (KATO)

Other Resources

Initial Committers

Steve Poole spoole at uk dot ibm dot com

Stuart Monteith monteith at uk dot ibm dot com

Carmine Cristello cristallo at uk dot ibm dot com

Richard Cole rich dot cole at uk dot ibm dot com

Christian Glatschke christian at glatschke dot com

Sonal Goyal sonalgoyal4 at gmail dot com

Affiliations

The initial committers listed are affiliated with IBM and self.

Champion

Geir Magnusson Jr has agreed to be Champion

Proposed Apache Sponsor

Incubator PMC

Nominated Mentors

Geir has volunteered to be a mentor.

Ant Elder

Robert Burrell Donkin