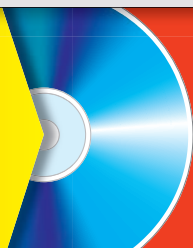


JavaTMmagazin

Java • Architekturen • SOA • Agile

www.javamagazin.de 

**Mit
CD**



Testversionen & more

- CAS Server 3.1.1
- JOSSO 1.6
- Eclipse Modeling Framework Project (EMF)
- Modeling Workflow Engine (MWE)
- Glassfish V2
- wingS 3.1

**PLUS
weitere
Top-Tools**

Infos auf S. 35

Specials

- JAX TV: Keynote von der W-JAX 2007



- Buchauszug: Hibernate – Praxisbuch für Entwickler

+ Bonus-CD für Abonnenten: Alle Ausgaben aus 2007!

Single Sign-on

SSO-Systeme auf dem Prüfstand

Test F...

dynaTrace
software

Offprint
by dynatrace

Systemarchitektur mit der UML
Kleine Anpassungen, großer Nutzen

→ **Get your wingS back!**

Framework für Ajax-Anwendungen

→ **Domänenspezifische Sprachen**

Erstellung einer textuellen DSL mit Xtext

→ **Software-Industrialisierung**

Java Banking Framework



Java Performance Tools, Part 2

Profiling, Diagnosis and Monitoring

■ BY MIRCO NOVAKOVIC AND MARC VAN DEN BOGAARD

The market offers different profiling-, diagnosis- and monitoring tools, some of them differ strongly from others concerning the functions. We will introduce some of them in this article, discuss the differences and give you information for your choice.

As soon as applications are used for mission critical processes, performance and availability are important non-functional requirements. Different tools are used for ensuring the performance during the complete lifecycle of an application. However, within the miscellaneous tool categories there are huge differences in functionality and appliance. If a tool works and how good it works will always depend on the requirements you have. The following examination of different tools is not supposed to determine a “winner” but describe different approaches and functions with examples to facilitate the choice and evaluation of performance tools.

In the first part of this article series about Java performance tools [1], the basics and technologies of nearly all tools mentioned have been described. The Java performance tools have been subdivided into three usage categories: profiler-, diagnosis- and monitoring tools. In the following, functions of and requirements for these tools are described and explained based on selected products. It is not our aim to compare the tools with each other and to show

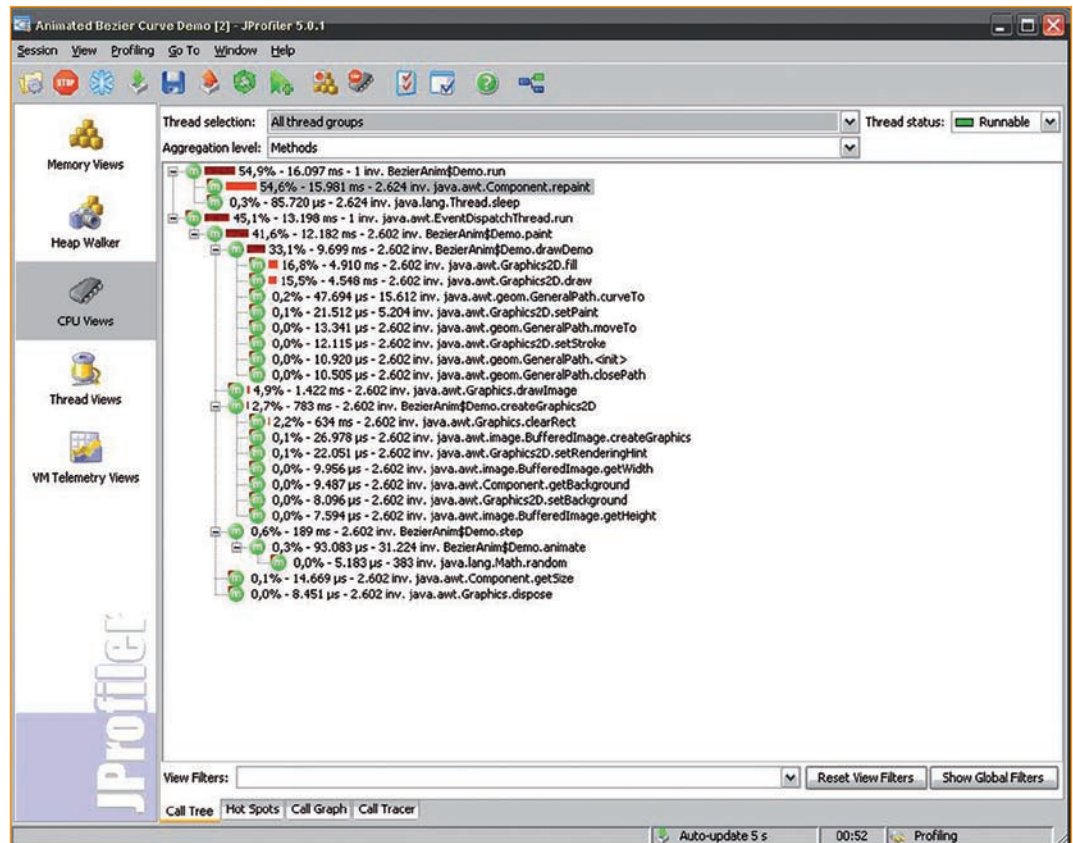
advantages or disadvantages, but to describe available functionalities of state-of-the-art performance tools to facilitate your choice.

Profiler

A profiler is a tool for developer to analyze performance- and stability issues. A profiler offers functionality to examine run time behavior and concurrency of an application as well as the storage behavior or concerning referenced objects and garbage collection. When analyzing the storage behavior, a huge overhead is created, caused by the profiler, which cannot be tolerated except in the development phase. In the area of performance analysis, profiler now work with byte code instrumentation and offer options for an overhead reduction with the help of filter functions. With these filter functions you can specify which methods, classes or packets should be used as instruments. Profiler usually work based on dedicated measurements and so-called snapshots. These snapshots contain all relevant measurement data, can be stored and analyzed via user interface of the profiler. A snapshot can be created in different ways:

- Manually via user interface of the profiler or via command line interface.
 - With the help of so-called triggers, trigger events which start or stop a measurement. A trigger can be the starting- or quitting-point of a method but also a threshold which is reached. Expanded functions of triggers such as certain call parameters for methods or the call frequency are useful functions to control measurements in detail. Triggers are not related to alerting functions as they are used for monitoring.
 - With the help of an API. Some profilers have their own API with which the functions can be controlled. This means that an arbitrary number of measuring points can be defined and the developer has maximum flexibility.
- but this also means the source code must be assembled accordingly for this variant.
- With the help of ant tasks. Some profilers offer ant tasks for automating the measurements which can be started and stopped with these ant tasks. Most of the time the tasks work in combination with the above mentioned triggers. Automating within a continuous integra-

Figure 1: JProfiler Call-Tree



tion [2] build process reduces the time for the performance data survey. It is also the most efficient way to establish the profiling within the development process.

- Some profiler also work in „always on“ mode which means that each profiling diagnosis and monitoring activity is recorded. The user can decide afterwards which data is to be stored.

Apart from these general functions, the integration into the development environment plays a substantial role when choosing a tool. Most profilers have their own user interface, but offer integration into IDEs so start configurations (e.g. for Eclipse) can be adopted. Another integration method is linking the sources. When a hotspot is found, a jump to the corresponding code will occur.

Open source profiler like Eclipse TPTP [3] or Netbeans Profiler [4], on the other hand, are plug-ins for their IDE and thus completely integrated.

Profiler offer different views for the run time analysis. The most popular are:

- the call tree, which depicts the methods called in a tree, according to their call hierarchy. This makes it easier to identify critical paths. Figure 1 shows a call tree in JProfiler [5].
- the call graph which depicts the calls in a directed graph. This allows a quick analysis of the critical path. The method list which displays all methods in a list so they can be sorted according to the hot-spots or examined for certain noticeable parameters.

Apart from the form of presentation, the captured data is particularly important. Most offer profiler data about the run time of methods and differentiate between clock time and CPU time. Clock time is the actually measured run time of the method and CPU time is the time in which the CPU has worked for this method. This means that clock time minus CPU time is the time in which the method waited for other resources. Apart from this data, many profiler offer data about the GC-time, the number of created objects, method parameters or additional information

about certain technologies. dynaTrace Diagnostics [6] for example uses the PurePath-technology which generally collects all data of a call path about a defined entry point. The PurePath is captured beyond the JVM-limits and thus enables performance measurements in distributed multi-level architectures.

Apart from Java, .NET is supported which means that interactions between these technologies can be analyzed. The PurePath itself is depicted as call tree, as you can see in figure 1. It contains on request, apart from the data mentioned above, information about method parameters and the return value, HTTP request parameters, header and session attribute values, SQL statement parameters or the size of serialized data of remote calls. With the help of so-called sensor packs, the captured data for certain technologies is defined and integrated into the tool. dynaTrace Diagnostics already offers a huge number of sensor packs for the different Java technologies and frameworks but also enables the creation of custom sensor packs.

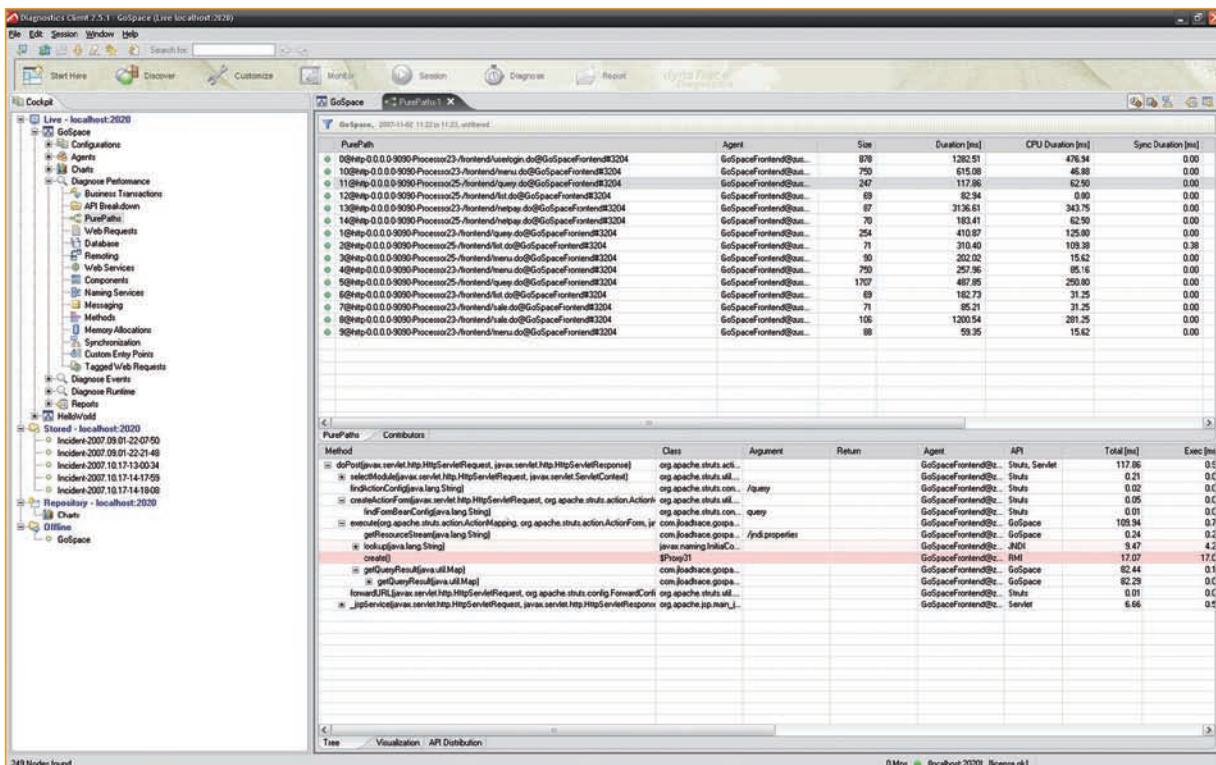


Figure 2:
dynaTrace
Diagnostics
PurePath

To analyze synchronization- and deadlock issues, you need information about the wait- and synchronizing times which have already have been described during the run time analysis. Some profilers also offer expanded analysis functions to analyze particularly these issues in detail. JProfiler for example offers a history overview of all threads and their status (runnable, waiting, blocked, Net I/O) over time. With the help of the current monitor usage view you can get a quick overview of the monitors which keep the

threads in certain classes. You can also see the threads which await the release from the monitor.

JProfiler shows, apart from the waiting threads and the threads to be blocked, traces which led to the synchronized resource. That way synchronizing issues and deadlocks can be identified quickly.

JXInsight [7] also has a timeline graph to identify so-called „concurrent workload patterns“. Figure 3 shows such a pattern. You can see two phases in

which the response time of the system is particularly bad from time to time. This is shown with the red bars in the top part of the graphics. In the lower part, the request including the updates (blue bars) and selects (yellow bars) are depicted in data base tables. It becomes apparent that the simultaneous access of multiple threads to one and the same data base resource led to table locks which explains the higher response times. Apart from the run time- and concurrency analysis, the analysis of the memory and garbage collection is an important profiler function. To identify so-called cycling objects (a high quantity of temporary objects which have a negative effect on the garbage collection), profiler can also record all objects which have been created during a measurement. The profiler generally differentiates between objects which have been cleared by the garbage collection and those which still are referenced in the memory.

Profiler such as JProbe [8] and JProfiler also offer the option to display allocation trees for the created objects. This means that the allocation hotspots in the code can be identified quickly. To identify Java memory leaks you need different functions.

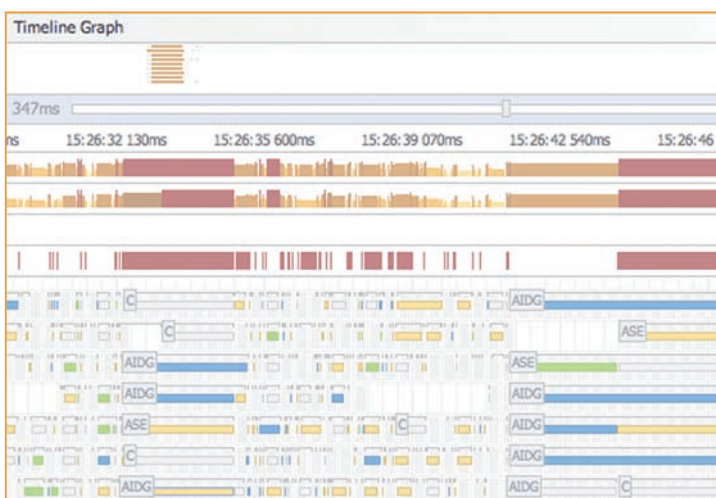


Figure 3: JXInsight
Timeline Graph

It is important that you can compare memory snapshots created in different points in time, to get a first hint which objects were added to the heap.

When the heap objects are captured, the overhead is enormous. This is why most tools offer the option to create snapshots which do not contain any information about references between the objects. With the help of these snapshots and the option to compare them you get a list of potential memory leak candidates. To analyze the exact cause of the memory leaks you need a heapdump which captures all heap objects including the references between the objects. Some tools also offer the option to capture the object content so you get information similar to the information in a debugger.

Apart from the memory leak analysis, you can also assess the object size. This is particularly helpful when you need exact data about session- or cache sizes. The overhead for creating a heapdump is very high which means that in productive environments an analysis is, most of the times, not possible.

As described in the first part of this article series, state-of-the-art JVMs offer the function to create automated heapdumps if an *OutOfMemoryError* occurs. JProbe and JProfiler offer the option to read-in these heapdumps (JProbe for IBM and JProfiler for Sun Hotspot) and to convert them into the profiler-specific format. That way current memory leaks can be analyzed with profiler functions. There are different methods and approaches for the identification of memory leaks. Most profiler support an automatic calculation of the path to the GC roots to identify the potential reference paths that cause a memory leak. JProbe offers Leak Doctor, a tool which tries to identify memory leaks based on different algorithms. With the help of the common parent analysis, the tool analyses e.g. if objects of the same type have common parent objects. This is for example the case if the objects are in a common cache and this still contains the references to the cache entries.

dynaTrace diagnostics shows object allocation in the described PurePath so

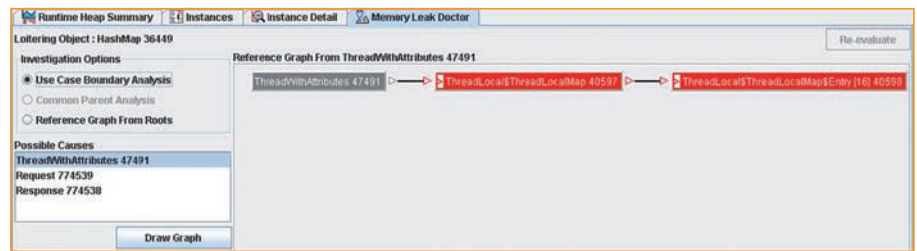


Figure 4: JProbe Leak Doctor

memory leaks can be attributed to single calls. The objects that are captured during this are defined via so-called memory sensors.

Diagnosis

In quality assurance, appliances are tested with the help of performance- and load test, so you can see whether they meet the requirements concerning response behavior, throughput, parallel users and stability. This test is called black box test as it is not possible to analyze the cause for a requirement which is not met. Diagnosis software bridges that gap and offers the option to analyze run time, concurrency and storing behavior under load. In contrast to profilers, these tools capture less detail and also include system metrics into the measurement. The overhead of these tools is a substantial factor for

getting exact load test results. Many software producers state overhead in per cent numbers, but these values are not easily transferable to your own environment. The overhead of a tool strongly depends on the set filters and the application to be measured.

If, for example, 80 % of the response time for a request is used in the data base and the byte code instrumentation reaches an overhead of 10%, then only 20% of the response time is affected by that – the evident overhead is thus only 2%. But if millions of methods are called and all captured, the overhead is considerably higher.

In some tools, the filters can be adjusted to the run time and activated. The dynamic byte code instrumentation of Java 5 is used for that. Re-starts of systems are minimized with this technology.

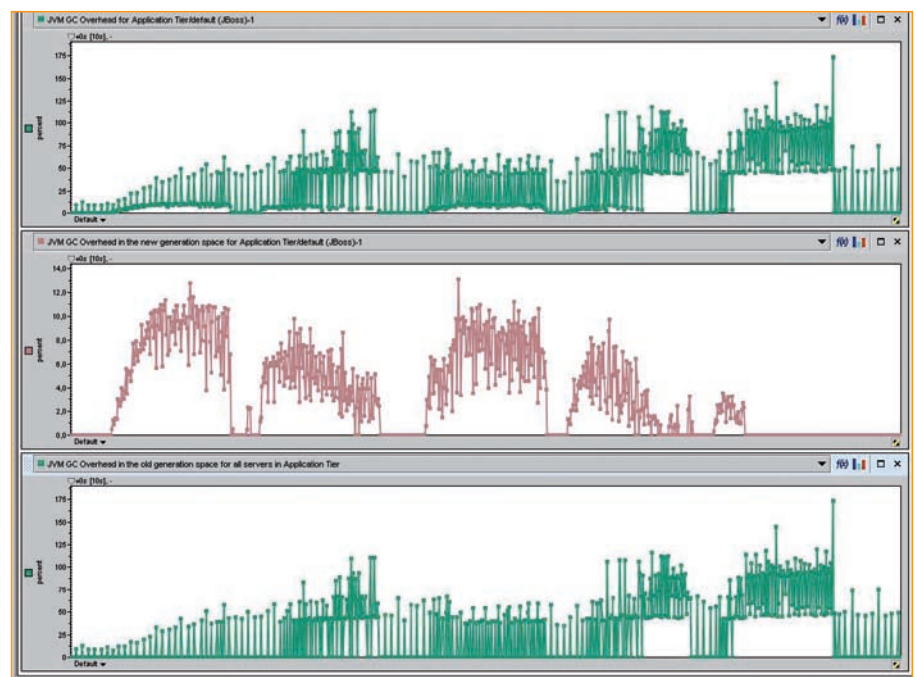


Figure 5: PerformaSure metrics browser



Figure 6: Foglight 5 Dashboard

Apart from the time overhead there is always a memory overhead. That means the measured data is kept in the memory by most of the tools and then sent to a remote data server, asynchronous and in regular intervals, to prevent a performance overhead. This is why a reduced performance overhead often means more memory usage. For these reasons it is difficult to compare diagnosis- and monitoring tools in the overhead area – in reality, only a proof of concept can show whether a tool is appropriate for a certain environment and for your own application.

Diagnosis tools work, in most cases, based on an agent technology.

This means that the systems to be tested are equipped with an agent that sends the data to a central data server. The agents are only data collectors when this happens, all processing logic occurs on the server. The data evaluation is realized with the help of an own client which connects to the data server and prepares the requested data graphically for an analysis. For the current distributed architectures it is important that diagnosis tools capture data across JVMs to be able to analyze call graphs connectedly. In a distributed EJB or web service environment, the data would otherwise have to be merged manually. There are two different approaches for capturing data in diagnosis tools.

Most tools work with statistical values and capture so-called samples of the incoming requests. The data is aggregated in certain time intervals and corresponding minimum-, maximum- and average values are generated. With the help of this procedure, critical requests, classes, SQLs and methods can be analyzed – single outliers however cannot be identified. Quest PerformaSure [9] is a typical example for a diagnosis tool based on this procedure.

The dynaTrace Diagnostics-PurePath-approach captures each incoming request by default. With this procedure single outliers can be identified. With the help of the context data of the PurePath (e.g. request parameters, method parameters) you can also analyze on which parameter the outlier possibly depends. In the case of a rule violation within a PurePath the whole corresponding PurePath or PurePaths of the following time period can be archived for analysis.

Apart from the instrumentation data, some more data plays an important role in diagnosis to analyze potential resource shortfalls. Among these is, for example, system data (e.g. CPU, memory, I/O), application server statistics (e.g. data source- and threadpool-utilization) and JVM-statistics (e.g. heap, GC, threads) are important to identify potential bottlenecks. PerformaSure, for instance, al-

lows showing the correlation between the collected metrics and the request response times. This means you can see the connection between the utilization of a data source and a bad response time.

An important function is the option to pass the results to the developers so the identified problems can be analyzed and corrected quickly. dynaTrace Diagnostics enables storing each PurePath. This can then be opened by the developers and linked directly to the sources with the help of an Eclipse plug-in. That means the problems with outliers or bad response times can be identified quickly. The advantage of this approach is that a problem can be analyzed without reproduction in the development environment. PerformaSure uses integration with JProbe for the communication with the developers. For critical classes or packets launcher files (parameter information) can be exported directly for JProbe. These files contain corresponding filters to analyze the problem in the profiler.

Monitoring

In the area of monitoring tools there are different approaches and architectures.

Generally we distinguish between tools which have an end-to-end-monitoring approach (Quest Foglight 5 [10], IBM IT-CAM [11]) and tools with which the Java platform is monitored as central integration component of current applications (CA/Wily Introscope [12], dynaTrace Diagnostics). End-to-End monitors do not only monitor Java but also different system components such as network infrastructure, data base, SAP, Siebel and mainframe applications under CICS or IMS. The advantage of this approach is that problems in the whole infrastructure can be identified. Foglight 5 makes it possible to write custom agents, additionally to the supported systems, and to integrate these agents into the monitoring infrastructure. That way, you can integrate e.g. self-developed C++ server into the monitoring process.

The Java monitoring tools concentrate on the Java components and integrate into monitoring infrastructures such as Tivoli or HP OpenView via the SNMP (simple network management proto-

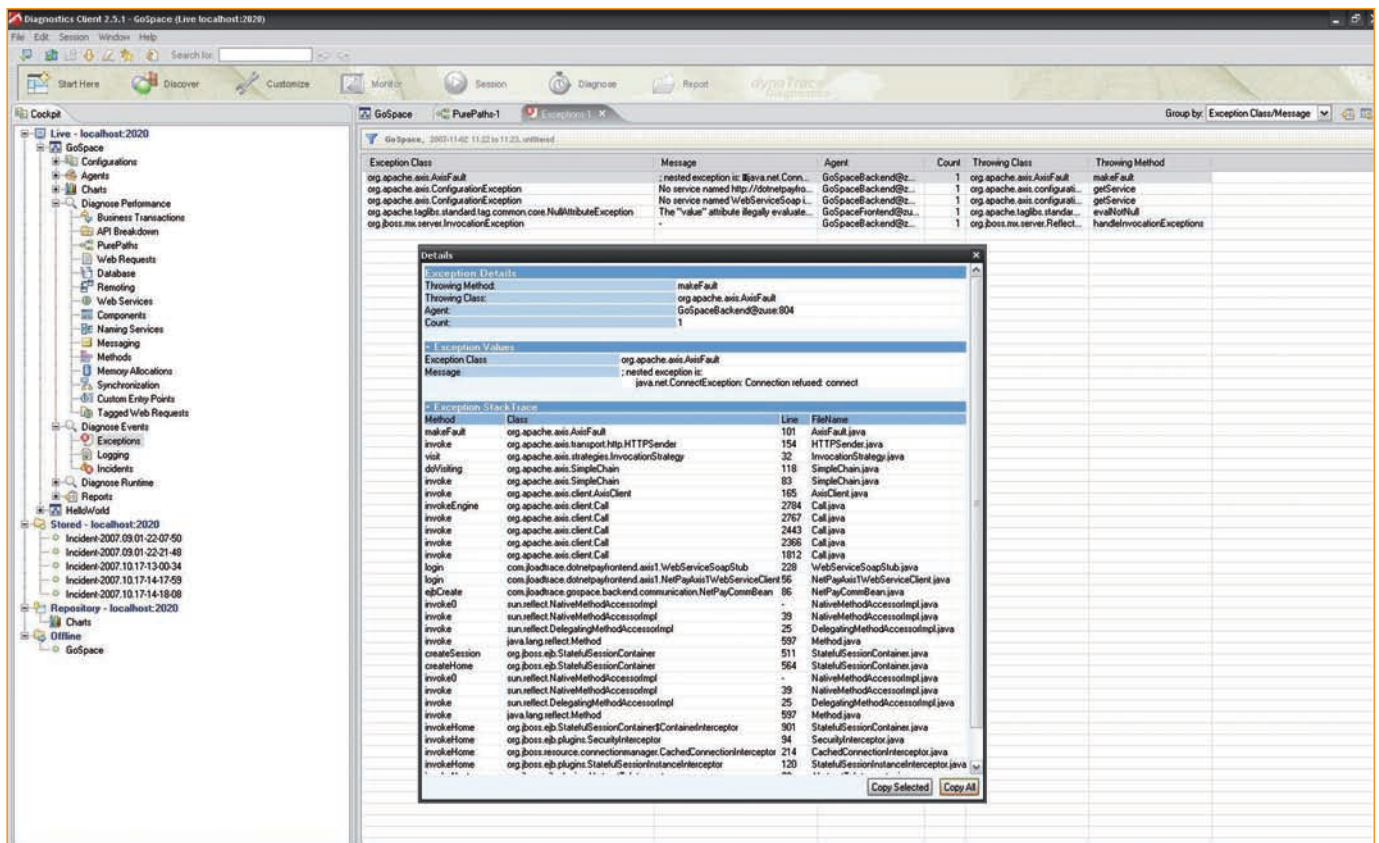


Figure 7: dynatrace diagnostics error analysis.

col) or provide their data via other interfaces.

Generally, Java monitoring solutions have to fulfill two primary requirements:

- Monitoring and logging of the operation
- Error identification and analysis.

Particularly in environments in which mission-critical applications are run, it is important to ensure a complete system monitoring. If a system downtime occurs, the administrator must be able to anticipate this if possible and must be informed immediately. Monitoring tools offer different functions for that. With the help of so-called dashboards the system status can be displayed visually. The dashboards function according to a drill-down principle. On the top level, the applications or servers are displayed and the system status is shown according to the traffic light principle (green: everything okay, yellow: critical, red: very critical or downtime). Based on the collected data, the status of a component can

be configured. You could, for instance, define that a system is green when the CPU usage is < 70%, yellow between 70% and 90% and red from 90% upwards. These so-called thresholds or incidents can, in most tools, also be defined based on response times of transactions or requests. The dashboards have a hierarchic structure so that the top level always adopts the critical status of a subordinated component. If a system goes to red or yellow, the administrator can drill down in the dashboards until he identifies the critical components. The dashboard technologies are substantially different: Wily Introscope uses a Java Client, whilst Foglight 5 offers a state-of-the-art, AJAX-based web surface (fig. 6). Both tools enable the display of custom dashboards per drag-and-drop. With the help of this function, “business dashboards” can be created which prepare data for the management. So-called derived metrics also enable the administrator in Foglight 5 to derive new metrics from different metrics, for instance for a

trend analysis. Based on metrics and incidents, most monitoring tools allow the control of actions. These actions can, for instance, send an e-mail or text message when a certain critical status is reached. In certain cases, an active reaction to the status is possible. That means an additional server could automatically be started or a less important application could be stopped, for instance, so an application with a higher priority can run without interruption.

In the area of monitoring, the duration of keeping the data is important, too. Monitoring data must normally be kept over months – this is the only way a trend analysis can actually work. Apart from that, SLAs (service level agreements) with the client or buyer are agreed on very often nowadays. Monitoring tools can monitor these SLAs and generate corresponding SLA reports. Data keeping cannot be managed durably without data aggregation in most cases – the data amount would simply become too big. This is why the monitoring data

is aggregated gradually. All details are stored for 24 hours and then aggregated within configurable time intervals. The older the data the bigger is this time interval. It is important that monitoring tools allow for a high flexibility in this area, so single values (for instance) or single applications can be treated in different ways.

Apart from these functions, some monitoring solutions specialized on Java also offer functions for the diagnosis of errors and performance-bottlenecks during operations.

dynaTrace Diagnostics, for example, captures the PurePath of each incoming request or Java- and .NET-call and delivers the described sensor data. This high degree of details reduces the time to error correction considerably. Each exception and each error log entry is captured by the sensors and stored including the corresponding PurePath.

This means that, when an error occurs during production, the corresponding PurePath can be given to the developer by the operators. The PurePath contains the whole call tree of the corresponding call including the request- and method parameters. This means that a reproduction of the error is easy. Without such a technology, such errors sometimes require days to be reproduced by the developer. The same goes for performance or resource bottlenecks – the PurePaths of the corresponding calls are all captured individually and can be analyzed online or offline.

CA Wily Customer Experience Manager and Foglight End User Management are examples for specialized solutions for monitoring to measure the response time behavior of an application from the user's point of view. Particularly these data deliver important information about the question whether mission-critical processes can be processed without problems. Both tools also offer the option to store the displayed websites of the users to provide them to the support specialists. With the help of created transactions it can be ensured that the application fulfils the requested SLAs from the user's point of view.

The different approaches of the tools show that the requirements are the most

important factor for choosing a monitoring tool. Very often, a single tool does not cover all requirements so you might need a combination of different monitoring tools.

Conclusion

This article has given you a short overview about the functions of profilers, performance, diagnosis- and monitoring tools. In environments in which mission-critical applications are developed, tested and operated, these tools are important. They help in the case of performance bottlenecks, stability problems or production errors to find a solution for the problem. First and foremost, the tools prevent unnecessary finger pointing between the individual organization units as the black box Java application becomes easier to comprehend for all involved.



Mirko Novakovic is manager of codecentric GmbH and specialist in the areas performance tuning, Java-EE-architectures and open source frameworks.



Marc van den Bogaard works as IT Consultant in open source-based Java EE-projects and consults customers with performance and stability problems.

■ Links & Literature

- [1] Mirko Novakovic, Marc van den Bogaard:
Java Performance Tools, Teil 1, in Java Magazin
12/2007
- [2] Continuous Integration: [martinfowler.com/
articles/continuousIntegration.html](http://martinfowler.com/articles/continuousIntegration.html)
- [3] Eclipse TPTP: www.eclipse.org/tptp/
- [4] Netbeans Profiler: [www.netbeans.org/products/
profiler/](http://www.netbeans.org/products/profiler/)
- [5] JProfiler: [www.ej-technologies.com/products/
jprofiler/overview.html](http://www.ej-technologies.com/products/jprofiler/overview.html)
- [6] dynaTrace Diagnostics: www.dynatrace.com
- [7] JInspired JXInsight: www.jinspired.com/
- [8] Quest JProbe: www.quest.com/jprobe/
- [9] PerformaSure: www.quest.com/performasure/
- [10] Quest Foglight: www.quest.com/foglight/
- [11] IBM Tivoli Composite Application Manager:
www.ibm.com/software/info/tivoli/itcam/de
- [12] CA Wily Introscope: [www.wilytech.com/
solutions/products/Introscope.html](http://www.wilytech.com/solutions/products/Introscope.html)

dynaTrace
software

Freistädter Straße 313
4040 Linz
Tel. +43 732 908 208
Fax +43 732 210 1000 08
office@dynatrace.com
www.dynatrace.com