Tech Notes

Performance Tuning Essentials for Java™

Manage Application Speed, Scalability and Reliability Throughout the Development Process

Al F. Mannarino, Embarcadero Technologies

August 2009

Corporate Headquarters
100 California Street, 12th Floor
San Francisco, California 94111

EMEA Headquarters
York House
18 York Road
Maidenhead, Berkshire
SL6 1SF, United Kingdom

Asia-Pacific Headquarters
L7. 313 La Trobe Street
Melbourne VIC 3000
Australia

WHY PERFORMANCE TUNING IS CRUCIAL FOR JAVA

In light of today's compressed development cycles, multi-tiered application architectures and complex technologies, many organizations are challenged to get reliable yet scalable enterprise Java applications out the door in a timely manner. Devoting a small amount of energy throughout the development process to identify, address and correct performance obstacles can dramatically lower the risks and costs associated with poorly performing applications over the life of the code.

Java performance tuning simply means optimizing your code for speed, reliability, scalability and maintainability. Producing truly scalable, lightning-fast Java SE and Java EE applications demands clarity of purpose and well-understood programming priorities. A major benefit of adopting regular performance tuning cycles is instantly seeing exactly which parts of your applications represent critical bottlenecks and which are behaving efficiently. Embarcadero® J Optimizer™ helps you do just that.

PERFORMANCE TUNING: A DEVELOPMENT BEST PRACTICE

A major strength of Java is its platform-independent byte-code approach and automatic handling of garbage collection. Unlike with C/C++, developers are able to focus on an application's business requirements, and are largely free from platform considerations.

Experienced developers, however, do not focus exclusively on application functionality. The reality is that below this level of abstraction, hard limitations of memory and processing power exist, as do the patterns and constraints of garbage collection, thread scheduling, and a host of other considerations managed by the Java Virtual Machine (JVM™) and the underlying operating system.

Successful developers —and productive development processes — incorporate regular use of performance analysis from the earliest stage of code creation throughout the development process, into QA testing, and beyond. Frequent, frontline checking and testing of small modules of code by the principal author is often the best way to assure that the Java applications produced will be fast, reliable, and scalable.

SEEING THE BIG PICTURE MEANS DIGGING DEEP

The latest generation of Java application servers brings increased memory capacity and processing power to the party. However, even with this and the common practice of throwing more hardware at the problem can you ever really overcome truly flawed code. A single buggy line ripples forward and can cause application-wide bottlenecks or can mysteriously trigger disastrous crashes once an application is in production.

The developer's tall challenge is to determine which part of a Java SE or Java EE application is causing a performance bottleneck or memory issue. The strength of Java and its various platforms is the high level of abstraction, re-use of objects, and insulation from layers of processing and system dependencies. But while encapsulation is great for shielding you from vast lower-level complexities, it also leaves few clues about where to focus your performance attention.

Tools are needed that extend your intuition and let you effortlessly see and understand how your Java application behaves when running. With the advent of highly abstracted, object-oriented languages such as Java, Stanford Computer Science Emeritus Professor Donald Knuth

raised the concern that programmers are in danger of losing touch with the factors determining whether their code will run and scale well saying, "At first you try to ignore the details of what's happening at the lower levels. But when you're debugging, you can't afford to be too compartmentalized. You can't afford to only see things at the highest level of abstraction."

PERFORMANCE TOOLS ALLOW YOU TO BE SMART AND EFFICIENT ABOUT OPTIMIZING

Knuth advises that developers need insight about what's going on below the surface if their code is to be scalable, reliable, and fast. J Optimizer is designed to efficiently give developers easy-to-understand, powerful views into the Java Virtual Machine, providing just this kind of insight.

A fundamental question is: "What are the priority performance issues for this module or application?" Tools like Embarcadero J Optimizer, specifically designed for Java performance tuning, offer an ideal way to answer this question—and be assured that your code improvements are informed and efficient. Without tools to help prioritize key Java trouble areas, you are likely to spend a lot of time micro-optimizing unimportant sections at the expense of the crucial optimization issues that actually drive your application's overall performance.

The goal ought to be for each member of the development team to be equipped with the tools to be *smart* about performance tuning each step of the way. Smart performance tuning will take place in the context of an application's overall business requirements. Some tiny performance issues simply may not warrant improvement efforts. Other important optimizing trade-offs will arise only when components are brought together, at which point an understanding of the overall architecture will guide modifications.

Tuning your code for speed and performance iteratively, as you develop and bring modules of code together, is the best way to minimize frantic troubleshooting sessions at the end of a project — or worse, in production, where even small problems are transformed into costly, complex challenges. The tuned applications delivered to QA and to customers will instead be lightweight, stable, scalable, and screamingly fast.

CONCLUSION: PERFORMANCE TUNING IS CRUCIAL FOR JAVA

Conceiving, designing, and testing your approach against performance goals *as you build* means more than just avoiding underperforming applications or even crashes. By being appropriately alert to how your code performs throughout the development process, you avoid expensive, disruptive late-stage fixes. Fast, scalable, high-performance code is a design imperative from the beginning. It is also a serious, regularly exercised element of the development process for each front-line developer (not a specialized skill for an isolated performance team).

The premise behind getting tools like the J Optimizer in the hands of each developer on the team is simple: Nobody understands a piece of code as well as the person who creates it, and nobody is better positioned than the author to make improvements in the logic and implementation of that code.

EMBARCADERO® J OPTIMIZER™ MAKES PERFORMANCE TUNING EASY

The 80:20 rule fits well for performance tuning. Eighty percent of an application's performance problems are usually caused by no more than 20 percent of the code. The pitfall that

development teams fall into is that they delay the identification of key problem areas until too late, leading to expensive and risky late-stage application redesign. Taking a proactive approach will preempt just this type of rushed, endgame rework. Adopting tools such as J Optimizer allows everyone in a development team to become more conscious of—and more conscientious about—the application's speed, scalability, and reliability throughout the development process.

As each routine or subsystem's basic functionality is established, the module's performance can be evaluated within the context of the applications overall business requirements. Clearly not every section of code will need to be labored over with equal intensity and certain key issues will emerge only as pieces of code are combined.

Whatever the situation, spotting and assessing the severity of issues throughout the development process avoids delays, problem escalations, and the kind of deep-rooted performance flaws that can sideline an entire project or create business disasters once an application is live with customers in a production environment.

About the Author

Al Mannarino is currently a Lead Systems Engineer and Evangelist for Embarcadero Technologies, Inc. Before joining Embarcadero, his last three years was spent at CodeGear from Borland and the prior five years was spent as a Lead Systems Engineer with Borland helping to sell ALM/SDO/Developer Tools solutions. Al has over 25 years of Software Development experience including OOAD and the responsibility for developing and deploying production applications. Prior to Borland, Al was an SE for Objectivity, Versant, Red Brick Systems, Information Builders, and an Electrical Engineer with Grumman Aerospace where he performed real application implementations on complex electrical-mechanical systems. Al has a BS in Electrical Engineering from Manhattan College.

Embarcadero Technologies, Inc. empowers application developers and database professionals with award-winning tools to design, build and run software applications in the environment they choose. With the acquisition of CodeGear in 2008, Embarcadero now serves more than three million professionals worldwide with tools that are both interoperable and integrated. From individual software vendors (ISVs) and developers to DBAs, database professionals and large enterprise teams, Embarcadero's tools are used in the most demanding vertical industries in 29 countries and by 90 of the Fortune 100. The company's flagship tools include: Change Manager™, Emabrcadero® RAD Studio, DBArtisan®, Delphi®, ER/Studio®, JBuilder® and Rapid SQL®. Founded in 1993, Embarcadero is headquartered in San Francisco, with offices located around the globe. For more information, visit www.embarcadero.com.