Title of Book – An eccentric looking at programming and software technology

by
Berlin Brown
01/28/2025 (v1)

Chapter One on Programming in Java

"If you want to know the past, look at your present. If you want to know the future, look at your present." -- Gautama Buddha

If you are software developer that mainly writes applications for the web; even if you aren't a Java/J2EE developer, you have probably heard the new meme circulating the web and online discussions. "Java is the new COBOL". Most blogs and technical writers regurgitate that statement and follow-up by saying that EJBs suck, and some companies are considering Ruby on Rails. I didn't really think much of these posts. I get the impression that a lot of these "managers" don't write any Java or COBOL code. They are really not highlighting the seriousness of the problem. COBOL development has remained stagnant for decades. Java may see the same problem in the future. The Pentagon/DOD has a trillion dollar COBOL problem because it is so difficult to change their ancient infrastructure. Anyone who is exposed to the Java world will become frustrated with some aspects of the environment. The beginner will certainly become frustrated with the amount of work it takes to perform simple tasks like opening a file or working with Swing to create a small GUI application and they will find it much easier to perform these tasks in a language like Python. The seasoned developer will find fault with Java's stability or the complexity of the various web frameworks (XML hell). The programming language designer will find fault with all aspects of Java. I believe that the "Java is the new COBOL" is a politically correct way for saying "Java sucks". I won't disagree with those statements. I am not overly partial to any technology, I just want to get work done and find the technology that makes that work as painless as possible. As an added bonus, I want to ensure that my time is spent on a project is worthwhile. What is worthwhile? If I had a choice, I would much rather work on or learn more about the software used to work with the Phoenix Mars Lander [1]. The lander was sent 422 millions miles and the robotic systems must operate autonomously, sending data and images back to Earth. The software used to build this software is probably more complicated than what is on the other side of the spectrum, for example MySpace or a MySpace application. Some software is more complex, more interesting, and generally more useful than other software. I am sure that the JPL team had to plan for almost any contingency from the physical complexities of the Mars environment, from the hardware components as well as within the Phoenix's internal software components. Not everyone can or would want to work at the JPL, creating robotic software. There are always going to be different software needs and different requirements, but the software development community should make an effort to plan for contingencies and seek out the best technical solutions for their users. Does a Java architecture provide the best solution? Does a COBOL one? I guess depends on the requirement.

What about actual COBOL developers? What do they think?

"The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence." -- Dijkstra, 1975 [22]

What about critical applications like in the banking industry? Use your imagination to come up with the possible requirements for web oriented banking software for internal and external applications. These applications might need to keep up with credit-card, or debit-card transactions for online payments. They might want to store personal profiles of the users. Users will need to login into, register for, pay for and maintain their accounts. The application has to be secure and follow all of the SEC guidelines. Depending on the company, you might have hundreds of thousands of users with a running database that has been collecting data for 40-50 years. Imagine the number of transactions that must be logged in order to run your business. This activity must be logged and must be accurate 24/7. I can only imagine if I logged into my SunTrust account and my bank statement said I had $40 when I really had $40,000. There are issues with these systems and errors due occur, but they are kept to a minimum. Imagine the requirements for the external application used by members of the public and then imagine the number of internal applications needed to maintain the public facing tools. You could have log-monitors of the logs. You could have sophisticated, custom build tools. You could have custom requirements document management systems. I remember laughing and reading a recent article about DreamHost's billing issue. "On January 15, 2008, the billing system was mistakenly used to bill users up to December 2008." That couldn't happen for all of the users at Bank of America or SunTrust.

COBOL's history is rooted in its creation in 1959 as a mainstay of business programming in the finance industry. COBOL was created in 1959 and is a mainstay of business programming in the finance industry. It is normally being used for the back-end and data processing work and various Java and .NET middleware libraries are being used to connect web interfaces to the running mainframe COBOL software. "In 1997, the Gartner Group reported that 80% of the world's business ran on COBOL with over 200 billion lines of code in existence and with an estimated 5 billion lines of new code annually.[4] IBM is one of the larger mainframe hardware vendors. So, if you hear about a mainframe shop, you can safely assume they are running a bunch of IBM hardware and software. There are still COBOL jobs available, there are still software and hardware vendors like IBM pushing their products. And it doesn't have to be an z/OS or AIX system. You can run SUSE linux on a virtualized LPAR (logical partition) or any other supported operating system all on the same machine. I don't understand any of the terminology and architecture of COBOL/Mainframe systems. Most COBOL (Common Business-Oriented Language) systems arenteract with large databases, that might include launching stored procedures and manipulating. A mainframe JCL (Job Control Language) script that will be placed on the queue and run as a job on the mainframe. It can compile and link a COBOL program and then execute the program.

The listing below is a brief snippet of COBOL code, without spending much time with COBOL you can get a glimpse at why it isn't something one would consider to be a beautiful language.

```
00000-MAINLINE.
OPEN INPUT JCL-IN
OPEN OUTPUT JCL-OUT
PERFORM 1000-READ-MOVE UNTIL EOF
CLOSE JCL-IN
CLOSE JCL-OUT
```

```
GOBACK.
1000-READ-MOVE.
IF NOT EOF
READ JCL-IN INTO WS-RECORD
AT END MOVE 'Y' TO WS-EOF
END-READ
MOVE WS-RECORD TO JCL-RECOUT
END-IF
```

Actually, these systems are fairly stable. Like I said before, a bank system has to run 24/7, collecting and manipulating data. In a J2EE/Linux/Unix environment, if the your JVM lock-ups or crashes due to memory leak issues or unforeseen NullpointerException bugs; normally you have to call someone, identify the problem and immediately get your server running again. A mainframe zSeries system may only experience a couple of minutes of downtime a year and all of your hardware sub-systems can be hot-swapped.

COBOL is a programming language with a long history, but why isn't it commonly used for modern, innovative web development? The answer becomes clear once you work with the language. A hybrid Java/COBOL developer once explained it well: "COBOL is too basic. There isn't much new development or modern methodologies for it, and it's not exciting to work with. COBOL lacks contemporary web development libraries and frameworks." From my conversations with COBOL developers, many share this sentiment—it's difficult to feel enthusiastic about the language. Even without building or reviewing COBOL applications, it's evident that the language has stagnated. There are no new books being written, no conferences or training programs, and few innovations in its methodologies or best practices. A COBOL program written a decade ago is functionally the same as one written today. In 2008, COBOL was already struggling to meet the growing demands of users. Modern users expect sophisticated web interfaces, GUI applications, and technologies like 3D gaming, none of which align with COBOL's capabilities. While COBOL is adequate for tasks like processing credit card transactions or updating database logs, it is not well-suited for building complex systems like a 3D game or software for the Phoenix Mars Lander.

What about Java, is Java the new COBOL?

I haven't taken a lot of the online discussions seriously on the "Java is the new COBOL" meme because of the way that a lot of technology leaders frame the statement. The question is, "Should Java be used for new development and applications?". If you are a die-hard Java fanboy and your answer is yes under all circumstances; then you are doing a disservice to your users and your career as a technical professional. You are moving in the right direction, if your answer is no or maybe; you are dedicated to looking at:

- The various software technologies and evaluating them with a rational, objective analysis on how the technology can benefit your user's needs
- The available developer resources (this may include only one developer)
- The overall future maintenance of your application

What is wrong with Java?

"One does not accumulate but eliminate. It is not daily increase but daily decrease. The height of cultivation always runs to simplicity." -- Bruce Lee

I won't add more to what has already been said about Java and J2EE, later I am going to post

some of my favorite "Java Sucks" quotes. You might also see some misinformation on the blogsphere with their use of the term Java. Some use it to describe the Java programming language, some use it describe the entire Java platform. Some people even take it a step further to include Java the programming language, the virtual machine runtime created by Sun or other software vendors, J2EE (Enterprise Edition, including EJBs, Servlet Spec, JMS, JTA, etc), or any libraries built with Java. In my blog post, I am talking about Java-everything (including third-party applications) unless I specifically target the "programming language" or "virtual machine", for example. From some people, if they say "Java Sucks", you can always respond with; "Well, Sun's HotSpot just-in-time compiler is used to achieve great runtime speed. And just a note, some mistakenly criticize Java's(the JVM) performance based on Sun's early Sun's implementation. In 2000, Sun released the HotSpot VM with just-in-time complication which including much improved performance, compared to the bytecode interpreter released in 1995. [12] And Java is still used in many complex production systems. Plus, you are not always tied to using the Java programming language to write your applications. Many other programming languages run on the JVM, including.

- JRuby (The Ruby language on the java virtual machine) [9]
- Jython (Python on the JVM)
- Scala (A multi-paradigm (OO and functional programming) strongly-typed programming language [8]
- ABCL (Common Lisp)
- Clojure (A lisp implementation on the JVM

Interesting quotes about Java

- "The programmers you'll be able to hire to work on a Java project won't be as smart as the ones you could get to work on a project written in Python....frankly, the fact that good hackers prefer Python to Java should tell you something about the relative merits of those languages." -- Paul Graham [16]
- "What in the world is application-level crap like checkPrintJobAccess() doing in the base language class library? There's all kinds of special-case abstraction-breaking garbage like this. ", "Interfaces seem a huge, cheesy copout for avoiding multiple inheritance; they really seem like they were grafted on as an afterthought. Maybe there's a good reason for them being the way they are, but I don't see it; it looks like they were just looking for a way to multiply-inherit methods without allowing call-next-method and without allowing instance variables?" -- jwz on the java language [17]
- "Scala is a far superior language to Java...The status of “Java” has been artificially elevated by its groupthink members such that any attack or suggestion of inferiority is reviled, even if the suggestion is perfectly rational and able to be supported with evidence" -- Tony Morris, Offending Religiosity [18]
- "Static typing is limiting", "Arrays are broken", "A double-whammy when combined with static typing. There is this weird distinction between object types and primitive types", "Instead of adding expressive features to the language, the Java designers decided to focus on promoting code duplication, aka design patterns instead". "No REPL". -- Slava Pestov (November 21, 2004) [19]
- "WebWork, Tapestry, Spring, and pretty much every non-Struts framework users scoff and laugh at JSF. It’s ugly, it’s not intuitive, and it is hellbent on the Microsoft style approach; fuck the users and force them to use clever tools." -- Hani Suleiman, author of the BileBlog

- "Deployment is still the painful arduous hope-and-pray-you-get-a-helpful-hint process we've all come to know and love. The smallest hiccup will cause the server to flail about wildly, spewing error messages with a frequency that is only matched by the number of new pointless codehaus projects every week. The error message themselves are as delightfully whimsical as they've always been. Random ObjectNames, with tantalisingly empty 'I depend on:' proclamations. The 'Depends on me' of course still impressively manages to insist that ejb's within a module depend on a seemingly random other ejb in a far flung module that has absolutely no dependencies to anything. Hot redeployment of apps that failed to start up doesn't work worth crap, the stacktraces seem longer, and the whole thing just makes you run to your nearest j2ee vendor and beg them to let you fling cash at them just to stop the pain." -- Hani Suleiman on JBoss
- "Java desktop apps succeed only in niches where UI design and usability don't matter: development tools and enterprise software. Programmers expect things to be crude and complicated... and the poor users of enterprise software don't have a choice..." [27]

XML Configuration Hell

XML and property configuration hell does not plague all Java development, this is one misconception that permeates the online discussions. If you aren't writing web applications, you can certainly avoid XML oriented frameworks. And if you are working with web applications, you can use lighter frameworks like Wicket. One of Wicket's selling points is a "refreshing lack of XML". But there is a certainly a tendency with Java web frameworks to be configured by needlessly complex XML configuration files. You can attribute some of this universal standard to Sun's default J2EE XML configurations. The J2EE web.xml, application.xml, WEB-INF/*. One or two simple system XML configurations are not bad. But, you start to enter XML hell when you want to combine technologies. By default, you have to deploy a web.xml, application.xml. If you are using and combining common frameworks like Struts2 MVC, Hibernate ORM and Spring2, Log4j, JSF, Google Web Toolkit, Axis you could have one to many different configuration files per framework. Essentially, you could have a minimum of five or six different configuration files to setup even the simplest web application (if you mistakenly use those frameworks). The practice of over-engineering every solution is very prevalent in the Java community, this is why some may consider Java to be the new COBOL. COBOL was so stuck to its niche that it couldn't really evolve to match the ever changing software world. It wouldn't make sense to provide COBOL web portals or portable GUI applications even though the back-end systems were originally developed in COBOL. Java is falling into the same trap. Why should a developer have to waste time worrying about five to six hundred line configuration files, on top of a poorly designed, inconsistent Java API when web frameworks like Rails or Django can accomplish similar tasks in twenty or thirty lines of code? More code means that more things can go wrong.

Web 2.0 and New Requirements

Grace Hopper, the author of COBOL, helped standardize programming languages used by the Navy to integrate incompatible embedded systems. Hopper helped build many of the Navy's onboard computers; "Hopper was tasked with developing a program for shipboard ADP equipment support, building compilers for shipboard minicomputers was a major part of her workload". [15] Web development is not the end all of software development, but a lot companies understand the benefits

and safety net associated with this ubiquitous style of computing. And a lot of companies and users don't just want a basic MVC-1 database driven application. They want more, they want applications that interoperate with other successful services. For example, the popular web application Yelp.com [14] contains user-driven restaurant and business reviews in and around large cities. The application also links the location of these businesses with google maps. You can pinpoint on a particular restaurant and if you don't like that restaurant, you can highlight similar restaurants in the area on the map.

These are the new wave of applications that users are wanting. They have seen it done and they want more of it. Is Java capable of keeping up with new requirements? Are other languages and platforms better and easier to use to meet these software demands? Here are just a couple of applications that are being developed in the Web 2.0 world. And yes, the term Web 2.0 doesn't have any real meaning; it is yet another tech buzzword created by the consultants and trade magazines to describe trendy software technology. Essentially, these software problems are not complicated and certainly not new.

- Ajax and JS Widget Support
- Wiki Sites
- Web Forums
- Portlets
- Mashups
- Social Networking and User Driven Applications
- Web Services
- Semantic Web (E.g. FOAF)
- Rich Internet Applications
- Cloud Computing
- Clustered Applications

GASP Where are the libraries?

Building modern web applications around the Java platform is skirting on the edges of web-development, programming, and network administration. I don't think the term Java programmer is an accurate term to describe a Java developer's functions and workload. Java developer's aren't always programming to put together an application for the user. I was once tasked with converting HTML output to PDF documents. Most of the work involved finding good opensource libraries that supported at least HTML4 tags, XHTML and CSS. Most of the libraries required an input HTML string and streamed the output PDF to a file or some other medium. There was a requirement and a need; I researched various tools and integrated the library with our web application. I didn't do much programming. A lot of Java work follows a similar work flow. Traditional programming is not always required. How do you add Lucene full-text search to your application? Add Lucene to your classpath, come up with strategies for indexing your documents, provide text edit box for user's to query the system and you are essentially done. Once again, programming is not required for you to add this functionality. Let me say it another way; imagine if the Lucene library wasn't available and you still needed to provide full text search. If you wanted to build a search library, more programming work will be involved building the API then using an existing one.

What about the other Programming Language Alternatives? Could you use more than one General Purpose Language in your Architecture?

(Actually, to say languages like Lisp, Scala and Haskell are alternatives to Java is kind of an insult to these technologies. Java is really a corporation inspired alternative to existing software systems that garnished a lot of momentum over the last ten years.)

One thing I never understand about the Java group-think community is the fear of integrating new programming languages into their environment. At a minimum, most Java web developers have to work with Javascript, XML/XSLT, HTML/XHTML, SQL and Java itself, sometimes even C/C++. Designing an application with one or more general purpose languages is not totally a foreign concept. Twitter attempted to solve some of their scalability issues by integrating their Ruby on Rails web platform with Ejabberd Jabber/XMPP server as their instant messaging platform. Erlang is the most widely used programming language that has been known to use a message-passing concurrency model. [21] Erlang is strong at parallel, distributed applications. If your requirements change or you see an increase in demand that your current set of resources can't meet, then shouldn't investigate other options. Other programming languages? Here are just a dozen or so stable languages that have vibrant developer communities and capable libraries that can even be used for enterprise development:
Popular and Mature Modern Programming Languages

- Python/Jython
- Ruby/JRuby
- Erlang
- Haskell
- Scala
- Clojure
- Smalltalk/Seaside
- Lisp (Common Lisp/Scheme; SBCL, CLisp, PLT Scheme)
- Factor
- OCaml
- **F#**

The requirements for web development are not going to change and will increasingly get more complex. In defense of the Java developer, he may believe that his only option is to architect his application with a J2EE platform because the application server may have all of the libraries that he or she needs to complete his requirements. Normally, this is not the case. He probably doesn't need JCA, JMS or all of JAAS or EJBs. Yet, a compliant J2EE server will provide all of this whether you need it or not. But, if the enterprise developer is considering new languages and platforms, he might need a minimum set of libraries. If we go back to the requirements of the online banking software; you aren't going to consider a web platform that doesn't support HTTP over SSL. And you probably don't have the resources to implement SSL yourself. Most modern languages have some of the features listed below. They are mature and have been a part of the language for years. If the language doesn't have the particular feature, there is nothing preventing you from using a combination of general purpose languages in your architecture.

Some Typical Enterprise Requirements

- Mature IDE (E.g. Eclipse, RAD, IntelliJ IDEA)
- Unicode Support

- Stable XML Parsing Libraries
- Database drivers (E.g. JDBC)
- SSL support and Security platforms (E.g. JAAS)
- Email Libraries
- MVC Web Frameworks
- (Soap, REST, RPC) Webservices libraries
- Search (E.g. Lucene)
- Legacy communication libraries (E.g. JCA, CICS, POS
- Chart generation libraries
- Image manipulation libraries (E.g. JAI)
- Scalable - handle an increase in users or demand
- Good Speed Performance
- Technical Support?

Conclusion

So the question for the 2008 age developer is this; to all the millions of one language Java developers that are out there, are you going to continue to only work with the Java language and platform, five, ten, twenty years into the future? Are you going to consider the alternatives? Do you think your users might benefit from other technologies? And I am sorry but if the answer is no, fear of change is not the most adequate response. I have been reading a lot about Richard Stallman, the free software advocate. He has one central theme that he uses to defend his free software movement; free software is not about the developers, it is about the users. If you are dedicated to providing robust software, wanting to make a lot of money is not going to be one your goals. It really boils down to what you want to accomplish as a software developer. Is your goal to defend some programming language religious ideology or provide useful computing applications to users and the software community?

[15] Grace Hopper: Admiral of the Cyber Sea", Kathleen Broome Williams 2004

Footnotes

[23] A lot of developers have many suggestions for what and what not to teach in computer science. I recommend a course on the History of Computing/Software. I wouldn't have been so happy-go-lucky about Java and Fortran languages if I knew the entire history of software at the time.

[24] Why did you disable commenting? I am intentionally disabling comments. I don't want to get into the "Java Sucks", "Rails is awesome", "Lambda the Ultimate laughs at the Java community" argument. There are a couple of people that can bash Java and get away with. Slava Pestov (creator of Factor), Tony Morris (developer on ScalaCheck and works with Scala), Hani the Bile Blogger can get away with criticizing Java. They are fluent in the language and are great programmers. However, my blog post may encourage a bunch of wanna-be Rails fan-boys or Java drones to tell me how I have gone wrong. I had two goals with my post. First; I wanted to respond to the "Java is the new COBOL comments". Second; I wanted to understand why we use X or Y programming language. In some respects, I was talking to myself. Why Java? Why Lisp? Why Haskell? I know I probably have flaws in my posts and criticism is certainly welcome. Disabling comments will encourage healthy criticism. If you want to add something to the blog post or other comments, you can email me: berlin dot brown at gmail.com

Section Header - On Unit Testing, Java TDD for developers to write

I have read and I am reading about four or five posts a day about unit testing. It really has been a long time obsession for me. I have moved past the technical and practical considerations on unit testing frameworks and done with the debates with "should you use Junit or Mockito or Karma?" I am more interested in the psychology of unit testing, who does it, likes it, hates it? It really is one of those easy to learn, hard to master concepts. For example, many many may play chess when they are young and can end up being horribly chess players most of their life, I am part of that majority. Unfortunately, I have never played chess and sat down for hours and tried to master it. I never see the common patterns or have a developed end game. I mostly just play with a knowledge of the basic rules. Following good unit testing practices within your software development shop is a lot like playing chess. It is easy to learn and difficult master. Actually, there are a lot of big differences, chess is a game, chess is not coding, and people take their software development very seriously. So if you don't master unit testing, but are able to complete your job tasks, some might argue that is an acceptable risk in the world of software development. And why master chess or master unit testing? If developers are fine without unit testing, then why even suggest it. Some developers just don't want to invest the energy to master the practice. And in some development shops, there is no hard requirement to do so.

I am not going to convince you to write unit tests with this one post, I will leave that up to software guru Martin Fowler and the people at ThoughtWorks who have written large tomes on the subject. But I will present my thoughts on why some developers won't write unit tests but why they should. Those developers and architects that do advocate unit testing generally fall into that category where they have written just enough unit tests to find it useful and they generally love the practice, they also encourage others to follow along. I am sort of in that camp, I have almost become religious about it. I can't imagine my real code without unit tests and I just feel guilty by only testing through manual functional testing.

Jeff Atwood of Coding Horror wrote a short blog post on the topic, "I Pity the Fool Who Doesn't Write Unit Tests". Here is the one blurb that stuck out for me, "Even if you only agree with a quarter of the items on that list-- and I'd say at least half of them are true in my experience-- that is a huge step forward for software developers". And this one, "It's more fun to code with them than without". That is the essence of this unit testing religion, we can't force it on developers and we can't force developers to write unit tests only a certain way. I and many others don't believe in the practice of 100% coverage. You will rarely get there anyway, depending on the project or company. Some will argue that you shouldn't break the rule on non-determinism and this is a big one. Basically, the unit test should return the same output every time you run the test. You should avoid breaking this rule for unit tests but you can still write and add automated integration tests to your suite and not waste time, combine a collection of unit tests and integration tests. A simple integration test might test connecting to your REST microservice and validating the HTTP status code. At that point, your test moves into the integration testing category. If you connect to the database, run a particular SQL statement and validate data model returned from the SQL invocation, then your test is basically integration. Both scenarios are not units are non-deterministic but I would still consider them to be useful. Also, as a start for new developers getting familiar with unit testing, writing integration tests may be more familiar to them than decomposing or refactoring their code for a real unit test. There is a benefit in database or HTTP integration tests, you can add them to a test suite and run them in a automated form after a code change and after a build. Even bad tests can be useful.

Misko Hevery is creator of one of the most popular JavaScript frameworks to emerge in the last couple of years. It is a Google project that he started working as an Agile Coach. As he puts it, he wants

to maintain the high level of automated testing culture at Google. Most of his published articles are not about AngularJs but on the benefits of automated testing. I can only imagine that he developed the MVC JavaScript framework because the old crop of frameworks were a pain to work with for developers. They were not testable.
I have given my advocacy speech on unit testing, but how do I use it, what practices do I follow?

- For every piece of new code, I formulate a unit test case. New code could include my model structure or interface into my Java services. This is critical, unit testing encourages you to write testable early code. Meaning, I try to use interfaces and abstract classes which allow me to inject mock objects early in the development process.
- For local development, I can build, write code, write and update my unit tests and then run the automated suite of tests. The key part is re-running the test suite. Normally I want my unit tests to pass, if they don't pass then I can look at my code and refactor. Also, the code I write today, I can run a year from now, I should expect the same result.
- As you are writing your unit tests. Have fun, this is not production code, the unit tests don't run in production, you can test input as little or as much as possible.
- I try to avoid unit tests around code that doesn't do anything. Write unit tests around your modules that have some kind of behavior. We shouldn't write model POJO code with setters and getters, but there is no reason to test a setter method. It is more fun to code around the real functionality.
- Writing unit tests also encourages the developer to write testable code
- Write Java code that doesn't use static methods or variables. Imagine that, try writing code that doesn't make use of the static keyword. Why would you do this? Static, class level routines are procedural and inherently hard to test. You can override their functionality, they are completely class level.
- Writing unit tests encourages refactoring. Some refactoring may include the use of OOP techniques. Use interfaces and abstract classes.
- Use a DI/Dependency Injection framework like AngularJS (yea I called AngularJS DI), Spring or Guice. DI frameworks encourages the container to create new objects for you. Managing objects on your own and using the 'new' operator encourages untestable code.

In Summary, see what Jeff Atwood, Martin Fowler and Misko Hevery have said about Unit Testing. And we pity the fool that don't do it.

JVM Notebook and Programming

JVM Notebook: Basic Clojure, Java and JVM Language performance

"Measure, don't guess. The primary goal of all performance tuning exercises should be maximize the end user experience given the resource constraints." [1]

"It isn't so much a "farewell to the J" as an expansion of the platform opportunities Java provides. Sun's investment to power ongoing development of JRuby and Jython broadens the range and reach of Java, as a whole." -- Rick Ross

Overview and JVM Languages

One of the exciting trends to recently emerge from the Java community is the concept of the JVM language. These technologies are all that you would expect them to be. They are implementations of languages that run on the Java Virtual Machine. Some are newly created and some are based on existing, more mature languages. JRuby, Jython are two JVM languages based on CRuby and CPython.

Groovy, Scala, Clojure are three completely new JVM languages that were created to add new language features that weren't supported by the core Java language. Some must or can be compiled. Some run without compilation. You can easily compile Scala code to Java bytecode. Clojure also allows this feature (ahead of time compilation). Clojure and JRuby code can also run without having be explicitly compiled. You can interact with the language. In most cases and with most JVM languages, you have full access to existing libraries that were written in pure Java. And normally you can access existing JVM language code from Java (known as Java interoperability). In most cases, it is easier to access Java calls from the JVM language than it is to call the language code from Java. It really depends on the language. In the snippet below, there is a call to the System static method, 'nanoTime'. Simply invoke the system like you would from pure Java.

```
require 'java'
def helloWorldBaseline
# Run the application
start1 = java.lang.System.nanoTime()
arr = Array.new
(1..1000000).each {
res = 2.2 * 4.1 * 5.4
}
end1 = java.lang.System.nanoTime()
diff = (end1 - start1) * 1e-6
puts "Elapsed Time: #{diff} ms"
end
```

Dynamic Compilation - "Clojure is a compiled language, so one might wonder when you have to run the compiler. You don't. Anything you enter into the REPL or load using load-file is automatically compiled to JVM bytecode on the fly. Compiling ahead-of-time is also possible, but not required" -- Clojure Docs on being Dynamic.

On Performance

"If you want to measure something, then don’t measure other shit." -- Zed Shaw [3]

For the more popular JVM languages, like JRuby and Jython, there isn't much of a difference between running code in their respective C implementations. JRuby is especially well known for being very portable. With JRuby release 1.3.1, JRuby is compatible with CRuby 1.8.6. Jython 2.5.0 was released last month and brings the level of compatibility to CPython versions 2.5. Django and other popular CPython based frameworks are able to work with Jython. You may be wondering, if the Java Virtual Machine language is compatible with the C language, native implementation, is there a loss in performance when running on the Java Virtual Machine? Is there a major loss in performance? That is this purpose of this document, how much time does it take for a particular piece of code to run in JVM language? How long does it take to run similar routines using pure Java code? I want to make it clear, you will not find a large scientific benchmark run under clean room like conditions. I want to present a simple set of routines and how long it took to run. How long did the Clojure code run? How long did the Scala code run? Basically, I want to present the code and how long each test ran, but I don't want to claim that anyone language or piece of code is faster or slower based on these tests. You could say that most of the pure Java code ran faster. Most of the time, that is what happened after running these tests.

But there is too much confounding in my tests. Like Zed Shaw said, "If you want to measure something, then don't measure other shit." [3] There is a lot of stuff in my tests to not make these an official comparison. There is a lot of confounding. But, here is the code, here is how long it took to run? It be relevant in more common tests like a Project Euler problem. Project Euler is a website that

contains math problems intended to be solved with computer programs. In Project Euler problem number one, I write a program in Clojure and then in Java. They both run on the JVM and the same value is returned. What was the execution time for each program? Simple tests, simple results.

When working with JVM languages and possible performance bottlenecks, you want to consider execution time, but you also want to look at the garbage collector and heap memory usage. Garbage collection is an expensive operation. It won't take a minute to run a garbage collect, but it will take cpu cycles away from your application. JVM code runs in a protected environment, the garbage collector provides automatic memory management and normally protects you from improper memory use. And the garbage collector attempts to free up memory that is no longer needed. You can't normally control when the JVM runs garbage collection and certainly don't want force it. But if you monitor your application, you can identify memory leaks or other problems that might cause performance bottlenecks. It will normally be evident where there is a problem. If you see too many garbage collects within a very short period of time and your total available memory is maxed out, you might eventually encounter an out of memory error. In a long running server environment, most of your performance issues might be alleviated if you look at proper heap memory use. Is your code forcing too many garbage collections within a short period of time? Are you creating too many large objects and holding on to them for too long? In performance tuning your application, there are many things to consider. It may not just be improving a particular algorithm. Consider heap memory and object allocation as well. For most of the tests, there are performance stats, memory stats and other garbage collection statistics.

On Clojure

Clojure is a Lisp dialect created in 2007 by Rich Hickey. It recently reached a 1.0 release and has a large following of Java programmers and Common Lisp developers. Since the Dec 2008 release, the Clojure library has received 18,000 downloads (based on Google Code Stats). Clojure is a dynamically (dynamic/strong) typed language, supports lazy sequences, immutable data structures, macros, and functions as first class objects [4]. Clojure is a functional language just like Common Lisp is a functional language. It is a Lisp dialect so that includes the fully parenthesized syntax. Most syntax will include a function or macro call with arguments or no arguments enclosed by a left and right parenthesis.

Syntax

Most newcomers to a Lisp dialect may get distracted by the parentheses, the symbolic expressions. They can seem daunting if you are more familiar to a language like C++ or Java. But, this actually is one of the major benefits of Lisp. The simple syntax, functional call, args very much resemble how the compiler or parser will interpret the code. Simple is good. Simple is fast, especially to the machine. It can also benefit the developer because you aren't overburdened with a bunch of syntax to memorize.

It also really helps to have a great editor like Emacs. Emacs is built with its own Lisp dialect, Emacs Lisp. So, Clojure syntax is not too foreign to Emacs. You will need to download the Clojure Emacs Mode and you want to add Slime integration.

Here is a snippet of Clojure code. Just focus on the left parenthesis and the token adjacent to the character. The token, function or macro call and the left parenthesis.

Lisp does not normally get more complicated than the parenthesis tokens and defining function

bodies. A function/macro call and arguments. The arguments are normally separated by a variable number of spaces and may include calling another routine. That is the essence of functional programming. You have functions that return some value and can call other functions. You don't have to worry about Object creation syntax, for loop syntax, anonymous inner classes or things that you might encounter with Java. Here is some sample Java code. Look at all the tokens that part of the language.

With this document, I tried not to position one language as better or worse than the other. Each technology that I mention has advantages and disadvantages for writing software. Each tool may give the developer productivity gains and some developers may never get used to changing to a new syntax, never truly realizing some of the intended benefits that the language has to offer. You will have to evaluate these languages (or not) on your own and make. I merely try to point out some of the similarities and some of the differences.

Lisp certainly suffers from that old adage, "easy to learn, may take a lifetime to master". It is easy to write Lisp code, it may take time to write readable, solid, idiomatic Lisp code. I am sure I will get many comments on how to write more idiomatic Clojure code even for these rudimentary examples. Luckily, Clojure has many of the functions that you will encounter with most other Lisp dialects. In Java or C++, you may be accustomed to the 'for loop' syntax. Here is the Clojure macro for building a list of elements.

```
;; Build a collection of random numbers
(for [_ (range 100) ] (.nextInt random))
```

For Clojure API Documentation: "List comprehension. Takes a vector of one or more binding-form/collection-expr pairs, each followed by zero or more modifiers, and yields a lazy sequence of evaluations of expr. Collections are iterated in a nested fashion, rightmost fastest, and nested coll-exprs can refer to bindings created in prior binding-forms."

```
;; Macro definition for 'for'
(defmacro for
[seq-exprs body-expr]
(assert-args for
(vector? seq-exprs) "a vector for its binding"
(even? (count seq-exprs)) "an even number of forms in binding vector")
(let [to-groups (fn [seq-exprs]
(reduce (fn [groups [k v]]
(if (keyword? k)
(conj (pop groups) (conj (peek groups) [k v]))
(conj groups [k v])))
[] (partition 2 seq-exprs)))
err (fn [& msg] (throw (IllegalArgumentException. (apply str msg))))
emit-bind (fn emit-bind [[[bind expr & mod-pairs]
& [[_ next-expr] :as next-groups]]]
(let [giter (gensym "iter__")
gxs (gensym "s__")
do-mod (fn do-mod [[[k v :as pair] & etc]]
(cond
(= k :let) `(let ~v ~(do-mod etc))
(= k :while) `(when ~v ~(do-mod etc))
(= k :when) `(if ~v
~(do-mod etc)
(recur (rest ~gxs)))
(keyword? k) (err "Invalid 'for' keyword " k)
next-groups
`(let [iterys# ~(emit-bind next-groups)
fs# (seq (iterys# ~next-expr))]
```

```
(if fs#
(concat fs# (~giter (rest ~gxs)))
(recur (rest ~gxs))))
:else `(cons ~body-expr
(~giter (rest ~gxs)))))]
`(fn ~giter [~gxs]
(lazy-seq
(loop [~gxs ~gxs]
(when-first [~bind ~gxs]
~(do-mod mod-pairs)))))))]
`(let [iter# ~(emit-bind (to-groups seq-exprs))]
(iter# ~(second seq-exprs)))))
;; END OF MACRO (code from core.clj, clojure source);;
```

OK, that was a little daunting, but it is good to know the code for all of the Clojure source is freely available. This a yet another added benefit that you may not capitalize on, but all of the Clojure source is highly readable and lightweight. There are only 24k lines of Java code in the clojure.lang package. And only 7k lines of Clojure source in the core libray. I routinely make changes to the Clojure Java code to add trace logic. It took only 10 minutes to figure out where I needed to place my logic and then compile with my own Ant script. Imagine making similar changes to the Java API?

More on the Clojure, Java source for the performance tests

I used a simple format for the performance tests that I will present later in the document. I wrote or used existing code from a third-party and then ran the code. There may be many differences between the style of code. And the Clojure code may not be idiomatic. But it should be close. Here is the source for the Quick Sort, Clojure and Java:

```
;; Two versions of the Clojure Quick Sort:
(defn qsort
"Quick sort from rosetta code:
http://rosettacode.org/wiki/Quicksort#Clojure"
[[pvt & rs]]
(if pvt
`(~@(qsort (filter #(<  % pvt) rs))
~pvt
~@(qsort (filter #(>= % pvt) rs)))))
(defn qsort-2 [[pivot & xs]]
(when pivot
(let [smaller #(< % pivot)]
(lazy-cat (qsort-2 (filter smaller xs))
[pivot]
(qsort-2 (remove smaller xs))))))
(dotimes [x 4]
(println "i: " (int (Math/pow 10.0 x)))
(time (count (qsort (for [_ (range (int (Math/pow 10.0 x))) ]
(.nextInt random))))))
```

```
                          // JAVA VERSION OF QUICK SORT
public static final List quicksort(final List arr) {
if (arr.size() <= 1) {
return arr;
}
Integer pivot = (Integer) arr.get(0); //This pivot can change to get faster results
```

```
List less = new LinkedList();
List pivotList = new LinkedList();
List more = new LinkedList();

// Partition
for (Iterator it = arr.iterator(); it.hasNext();) {
Integer i = (Integer) it.next();
if (i.compareTo(pivot) < 0) {
less.add(i);
} else if (i.compareTo(pivot) > 0) {
more.add(i);
} else {
pivotList.add(i);
} // End of the if - else //

} // End of the for //

// Recursively sort sublists
less = quicksort(less);
more = quicksort(more);

// Concatenate results
less.addAll(pivotList);
less.addAll(more);
return less;
}
```

// END OF EXAMPLE

Performance of Test 11, Sorting

How did the various languages perform?

```
JavaTest   PerformanceTime(ms)
#####      ############
java11     1190.1847739999998
jruby11    5068.168914
clj11      11852.013362
```

JRuby and Scala Source, Scala Performance for the Simple Quick Sort

On Scala

Scala was created in 2001 and is currently at version 2.7.5 as of 6/3/2009. Scala is a stark contrast to Clojure. David Pollak had this to say about Scala's type system, "Put another way, Scala is statically typed with optional duck typing... the opposite of Objective-C et. al. with their duck typing with optional static typing." Clojure on the other hand is dynamic in the sense that you don't have to explicitly define a type whenever you need to use a particular object, but you can define the type specification using type hints.

```
;; Notice, second version uses the String type hint
(defn str-split-refl [s regex]
(vec (. s split regex)))
(defn str-split-fast [#^String s regex]
(vec (. s split regex)))
```

The above example presents two versions of a string split method, in Clojure. The second version use the String type hint to help the compiler determine the type for future calls in that function. On Scala's type system, "Scala is equipped with an expressive type system that enforces statically that abstractions are used in a safe and coherent manner" [6]

When would you use a JVM language? (Scala, Clojure, or just Java)

Why even investigate a JVM language? What is the point? A programming language is like any other tool or library used to create and interact with other software or hardware components? If you work in a web or J2EE environment, you might write SQL code, define CSS scripts, write Javascript code, write HTML/XHTML. It isn't uncommon for web application developers to write Java, sql, css, javascript or HTML code. Sometimes all within the same day, sometimes during different phases of a project. That doesn't take into account the number libraries that you must normally learn, understand and work with. So, NOT learning a new JVM programming language just on the premise that it is something different, isn't a valid reason for not using it. People have asked me, would you use Clojure, Java, or Scala? I have used all three for small GUI projects. I have used Scala for the the backend API where I don't need to make small changes. I used Clojure because of the dynamic nature of the language. I can make many, quick incremental changes without having any major impact on the Scala backend API. Take the Java api for example. Most of the core library is set in stone. The java.lang.String class hasn't changed much in over a decade. I see Scala being used for those type of rigid API requirements. This doesn't mean that Clojure couldn't be used for this purpose, it just means that is how I have used Scala and it just seemed to fit because of how easy it is to call Scala from Java (Java interoperability), also because of the nature of Scala's types. Here is just one example on how I used Clojure. The code snippet below contains valid Clojure code used to develop a small GUI application. If you just look at it without understanding the syntax, the code below almost looks like a general purposed configuration file. Here I can easily modify the layout of my GUI window, buttons without ever really getting any complex language details. I am just looking at the data required to change my layout.

A blogger from creativekarma.com provided a list of some of Scala's features:

```
* Conventional class types,
* Value class types,
* Nonnullable types,
* Monad types,
* Trait types,
* Singleton object types (procedural modules, utility classes, etc.),
* Compound types,
* Functional types,
* Case classes,
* Path-dependent types,
* Anonymous types,
* Self types,
* Type aliases,
* Generic types,
* Covariant generic types,
* Contravariant generic types,
* Bounded generic types,
* Abstract types,
* Existential types,
* Implicit types,
* Augmented types,
* View bounded types, and
```

```
* Structural types which allow a form of duck typing when all else fails." [8]

// Here is an Example Scala Quick Sort:
// http://en.literateprograms.org/Quicksort_(Scala)
// From scala home
def sort[A <% Ordered[A]](xs: List[A]):List[A] = {
if (xs.isEmpty || xs.tail.isEmpty) xs
else {
val pivot = xs( xs.length / 2)
// list partition
// initialize boxes
var lows: List[A] = Nil
var mids: List[A] = Nil
var highs: List[A] = Nil
for( val item <- xs) {
// classify item
if ( item == pivot) mids = item :: mids
else if (item < pivot) lows = item :: lows
else highs = item :: highs
}
// return sorted list appending chunks
sort(lows) ::: mids ::: sort(highs)
}
}
// Running the sort and time
def run(n:Int) = {
val start1 = java.lang.System.nanoTime()
//val l = repeat(n)(random.nextInt)
val l = initialize(n)
val l2 = sort(l)
val d = l2.length
val z = d + 1
val end1 = java.lang.System.nanoTime()
val diff = (end1 - start1) * 1e-6
println("Elapsed Time: " + diff + " ms " + z)
}
// End of scala example
```

JRuby Quick Sort

```
# JRuby Sort Example:
class Array
def quick_sort
return self if length <= 1
pivot = self[length / 2]
return (find_all { |i| i <  pivot }).quick_sort +
(find_all { |i| i == pivot }) +
(find_all { |i| i >  pivot }).quick_sort
end
end
def runTest1a(n)
# Run the application
start1 = java.lang.System.nanoTime()
arr = Array.new
(1..n).each {
arr << rand(100000)
}
res = arr.quick_sort
end1 = java.lang.System.nanoTime()
```

```
diff = (end1 - start1) * 1e-6
puts "Elapsed Time: #{diff} ms"
end
# End of Example
```

```
Test     PerformanceTime(ms)
#####    ###########
java11   1190.1847739999998
scala11  1989.283518
clj11    11852.013362
```

The line graph is a combination of Java, Scala, and Clojure execution times with an increasing number of random elements. Here are the results for the Clojure quick sort:

```
iterat   exectime(ms)
#####    #########
81       8.019583
243      16.82068
729      69.767482
2187     121.894408
6561     370.054832
19683    1096.553958
59049    3245.377605
177147   11852.013362
```

All Tests

I ran 14 different tests, ranging from a simple hello world application to the wider finder test. The figure below depicts some of the results.

```
CljTest PerformanceTime(ms)
##### ###########
hello1  145.996411    Simple hello world test, more than 10k iterations
build2  3474.550155   Build large and small set of random strings
str3 495.028198    Various string operations
euler4  341.872834    Euler problem number 1
euler5  116.512002    Euler problem number 2
str6 213.379223    Additional string operations
math7 1212.796289   Misc Math operations, including sqrt and other calls
coll8  16356.185166  Collection operation, simple word frequency test
frac9  354.23239     Mandelbrot fractal
sort11  438.477163    Sorting routines, mostly quick sort
vars12  9542.282043   Large and small object creation
wide13  907.491465    Wide finder project, 10k and smaller number of lines
```

```
JavaTest  PerformanceTime(ms)
##### ###########
hello1  27.110056         Simple hello world test, more than 10k iterations
build2  1940.5321219      Build large and small set of random strings
string3 1673.399982       Various string operations
euler4  23.101229         Euler problem number 1
euler5 1.5945639999999   Euler problem number 2
str6 436.737084        Additional string operations
math7 131.1656449998    Misc Math operations, including sqrt and other calls
coll8  972.775771        Collection operation, simple word frequency test
frac9  247.63905499998   Mandelbrot fractal
sort11  62.24679          Sorting routines, mostly quick sort
```

```
wide13  94.820562         Wide finder project, 10k and smaller number of lines
```

Source for Results and Code

All of the source is available (common document license/BSD license) through the google-code browsable SVN URL. Download the results, language source code or gnuplot source. Unfortunately, this data is spread out haphazardly throughout the repository.

On Garbage Collection

"If the garbage collector has become a bottleneck, you may wish to customize the generation sizes. Check the verbose garbage collector output, and then explore the sensitivity of your individual performance metric to the garbage collector parameters." -- Tuning Garbage Collection [4]

The default arrangement of generations (for all collectors with the exception of the throughput collector) looks something like this.

```
Here is some output from the garbage colection report:
...
[GC 1277K->390K(5056K), 0.0006050 secs]
[GC 1286K->399K(5056K), 0.0005540 secs]
[GC 1294K->407K(5056K), 0.0005580 secs]
[GC 1303K->416K(5056K), 0.0009580 secs]
[GC 1311K->424K(5056K), 0.0006540 secs]
[GC 1320K->431K(5056K), 0.0007520 secs]
[GC 1327K->431K(5056K), 0.0012980 secs]
...
...
```

"With the line below, indicate the combined size of live objects before and after garbage collection, respectively. After minor collections the count includes objects that aren't necessarily alive but can't be reclaimed, either because they are directly alive, or because they are within or referenced from the tenured generation. The number in parenthesis"

(776768K)(in the first line)

"is the total available space, not counting the space in the permanent generation, which is the total heap minus one of the survivor spaces. The minor collection took about a quarter of a second." [5]

The usage of the entire heap was reduced to about 51%

196016K->133633K(261184K)

The output is generated with the 'verbosegc' and other JVM options:
(Note: The duplicated 'verbosegc' options gives more information about the garbage collecting)

java -verbosegc -verbosegc -verbosegc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps CrashJVM -Xms24m -Xmx32m > gc_anal.txt

There are 1400 data points in the output file and in the image for the Clojure sort example. Each GC line in the gc_anal analysis file contains GC information about a minor garbage collection.

```
[GC [<collector>: <starting occupancy1> -> <ending occupancy1>, <pause time1> secs]
<starting occupancy3> -> <ending occupancy3>, <pause time3> secs]
```

"If the incremental collection of the tenured generation cannot be completed before tenured generation is full, a major collection occurs and is indicated in the verbosegc output by the 'MSC' in the output."

```
<starting occupancy1> is the occupancy of the young generation
before the collection
<ending occupancy1> is the occupancy of the young generation
after the collection
<starting occupancy3> is the occupancy of the entire heap
before the collection
<ending occupancy3> is the occupancy of the entire heap
after the collection
```

Our GC chart will only show 'used memory after the minor GC' and the total available memory after.

Here is the gnuplot representation of the used memory and total available after a minor garbage collection. There were 1400 data points in the clojure test. This is after running the quick sort code above.

```
Data Points used in graph:
used after gc | total after gc
...
...
26605 32576
26744 32576
27180 32576
27190 32576
27362 32576
27551 32576
line: 1421 -- 27684 32576
line: 1422 -- 27843 32576
```

The results for the pure Java version are presented below:

Scala results, 250+ minor garbage collections.

Large Objects

The quick sort example is not a good test to really push the garbage collector. Here is another test, one with Clojure and one with Java. I instantiate a large number of large objects and do the same for small objects. The garbage collector is better at handling smaller objects and consequently not as good at handling large objects (using the default GC rules). So, if you are looking at performance issues, you might look at how often large objects are being created and how long you are holding onto those objects. It is better to create many small objects and retain them for a short time than creating a few number large objects and retaining them for a long time. For example, I guess it is better to write short, static, utility methods and only create objects local to that method. "Large objects might be too big for Eden and will go directly to the old generation area; they will take longer to initialize (when setting fields to their default values such as null and zero); and they might cause fragmentation" [6]. A

large object is one that the size of the allocation of the large object within a memory heap exceeds a maximum contiguous free space within the Java heap.

I attempted to create a large object that had many fields with lots of String data. Using my approximate sizeof utility method, it looks like it takes 800kb to create this object.

```
private Object obj5b = StringUtils.concat(StringUtils.randomString(random, (14 *
kb)),
StringUtils.randomString(random, (14 * kb)));
```

With this code, there were 4700 minor garbage collections. It was interesting that total available memory never goes over 14mb. Here is the final console system output after running the program. The last line shows the used memory and total available JVM memory.

```
53.236: [GC 53.236: [DefNew: 542K->28K(576K), 0.0002510 secs] 2155K->1642K(4448K),
0.0002997 secs]
53.243: [GC 53.244: [DefNew: 540K->25K(576K), 0.0002716 secs] 2154K->1639K(4448K),
0.0003217 secs]
53.252: [GC 53.252: [DefNew: 537K->18K(576K), 0.0002698 secs] 2151K->1631K(4448K),
0.0003206 secs]
"Elapsed time: 9691.390453 msecs"
Done
Performing simple garbage collection cooldown
(used:2M/2M [4M,31M ])
(used:2M/2M [4M,31M ])
Format for the memory usage line (clojure source to create the line):
(defn *memory-usage* []
(str "(used:" (*used-memory-m*) "M/" (*free-memory-m*) "M [" (*total-memory-m*)
"M," (*max-memory-m*) "M ])"))
```

Shown below is the output for the Java test:

```
...
87.459: [GC 87.459: [DefNew: 441K->48K(640K), 0.0002922 secs] 5147K->4905K(8968K),
0.0003323 secs]
87.468: [GC 87.468: [DefNew: 624K->64K(640K), 0.0004192 secs] 5481K->5217K(8968K),
0.0004631 secs]
87.470: [GC 87.470: [DefNew: 213K->51K(640K), 0.0002372 secs] 5367K->5269K(8968K),
0.0002763 secs]
87.473: [GC 87.473: [DefNew: 627K->11K(640K), 0.0003707 secs] 5845K->5774K(8968K),
0.0004112 secs]
Elapsed time: 47982.607108 msecs
(used:5M/3M [8M,31M ])
(used:5M/3M [8M,31M ])
(used:5M/3M [8M,31M ])
Done
```

So far, we have only shown information on minor garbage collections. Here are the results of the major collects. The major collection are delineated by Tenured.

```
3.657: [GC 3.657: [DefNew: 568K->63K(576K), 0.0003546 secs]3.658: [Tenured: 7464K-
>506K(7508K), 0.0117550 secs] 7934K->506K(8084K), 0.0122011 secs]
```

You can see, major garbage collects seem to take more time and there are fewer of them in this result set.

```
...
1167 4000
```

```
1235 4000
1193 4000
line: 156 -- 1260 4000
```

And the Java results. There were 440+ GC collects.

Additional Tools

There is no shortage of tools for monitoring JVM performance. Mature, open and proprietary monitoring applications are available. Most of them focus garbage collection, heap memory, cpu monitoring, and method trace calls. The netbeans monitor, Sun's jconsole, jrat, and the Eclipse memory analysis tool are a few that I have used recently.

Looking at the top objects with the test examples

The chart report output from Eclipse's Memory Analysis Tool is depicted below. The overview contains the number of objects, number classes and classloader. The MAT tool also looks at potential memory leaks and the objects that are causing the problem, "biggest consumers" and a heap object histogram. To generate the data, we simply generate a heap dump and open the heap dump file.

Even with the 'hello world' test, you can see that many objects were created. And no, the Clojure hello world application that I provided is not the typical hello world app. I actually do some basic multiplication calculations over 1,000,000 iterations. Hello world in this case is a misnomer.

Here is comparable Java output. According to the Eclipse mat tool, almost 9000 objects were created.

Scala and JRuby Class Histogram Results:

Running JRat, Netbeans Profilers

Running JRat or Netbeans profilers is as simple as running with the appropriate JVM args allowing your application to run and then opening the files output after the program has exited.

I downloaded the shiftone-jrat.jar jar file, placed it my current working directory. And then added these args:

The screenshot above contains the jrat output after running the Clojure mandelbrot shootout application. I didn't want to analyze the results but there are a couple of stats worth taking a quick look at. The clojure.lang.Numbers.lt method had 87 million exits and a total method time of 142 milliseconds. The Numbers.ops method was called 367 million times.

The netbeans profiler contains similar profiling statistics as the jrat tool but the Netbeans profiler contains live, realtime results and a host of other metrics.

Additional Tests from Third Party (from AndyF):

Andy, a developer provided great Clojure performance information. I ran some of his Clojure source. Here are the runtime, garbage collection and memory results.

Source

I ran two types of shootout like tests, the threaded, non-threaded Mandelbrot test in Java and Clojure. Performance for these were similar to the all of the tests that we have run up to this point. The mandelbrot code seemed to perform slower than the 1:10 (java:clojure) speed ratio that we encountered. For example, the Java threaded code ran at '1700.062' ms. The Clojure version ran at 50998.30 ms.

MandelSiz	exectime.java(ms)	exectime.clojure(ms)
#####	#########	#####
100	11.515956	90.97791
200	22.154556	263.893038
300	42.036901	610.872097
400	70.4424729	982.452582
500	105.705857	1495.433005
600	151.296857	2091.412938
700	201.6435589	2843.495665
800	260.195961	3650.13744
1000	407.073079	5680.39932
1200	577.353532	8208.967304
1400	785.8052849	11126.994209
2000	1593.239278	22588.712673
2500	2487.326002	35210.526506
3000	3576.071694	50998.301209

The chart above contain the garbage collection results after running the Clojure mandelbrot shootout test.

Testing Environment

For this document, all of the tests were run with Java 5. (Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_11-b03)). Some tests were run with both the -client and -server options. The majority of tests were run with the -client option. Headius describes the -server option, "The -server option turns on the optimizing JIT along with a few other "server-class" settings.". Updates to the this document will include tests with Java 6.

Operating System and Hardware Environment:

- Java: Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_11-b03)
- Windows: Windows XP Professional Service Pack 2
- Hardware: Intel Core 2 CPU 6300 - 1.86ghZ, 3.49 GB of RAM

Virtual Machine Languages:

The focus of these tests are mostly to look at Clojure and how it compares to pure Java code. Clojure is included in all of the tests. There are only two tests for Scala.

- Clojure: 1.0.0/ 2009-05-04
- JRuby: 1.3.1 / 2009-06-15; 44 days ago
- Scala: 2.7.5/ 2009-06-03; 50 days ago

Resources

[1] - measure quote - - quote from the java performance tips. I have not found the original source of

this quote.
[2] - Rick Ross quote -
[3] - Zed Shaw on Statistics -
[4] - Clojure Home -
[5] - Tuning Garbage Collection -
[6] - On large vs small -
[7] - Scala's, static typing - http://en.wikipedia.org/wiki/Scala_(programming_language)#Static_typing
[8] -

Additional Links

Download or Browse All Source:

Section Header - Wolfram's Cellular Automata, A New Kind of Science and Example Squaring Rule

When most computer users upload a profile image from their desktop to Facebook's website they don't stop to think about the simple binary math rules that are fundamental to most digital devices. We realize that 4 gigabytes of RAM is more memory than 512 megabytes but we don't visualize the logic chips that are involved in an xor $0x100, eax operation for a 32-bit CISC processor. Software developers have to consider memory management or how a computer's operating system loads their programs into memory. They don't normally consider VHDL logic circuit designs, the data paths, arithmetic logic units or the millions of transistors that make up a modern CPU. Those low-level details have been intentionally hidden from the user application developer. The modern CPU may have changed dramatically over the last decade but at the heart of early digital computing were simple Boolean operations. These simple rules were combined together and logic replicated to load programs into memory and then execute. The rules that control most digital devices are based on elementary Boolean rules. Cellular automata has a similar bottom-up approach, rules consist of simple programs (as Stephen Wolfram calls them) that apply to a set of cells on a grid.

Conway's Game of Life cellular automaton is one of the most prominent examples of cellular automata theory. The one dimensional program consists of a cell grid typically with several dozen or

more rows and similar number of columns. Each cell on the grid has an on or off Boolean state. Every cell on the grid survives or dies to the next generation depending on the game of life rules. If there are too many neighbors surrounding a cell then the cell dies due to overcrowding. If there is only one neighbor cell, the base cell dies due to under-population. Activity on a particular cell is not interesting but when you run the entire system for many generations, a group of patterns begin to form.

More on Conway's Game of Life

Figure: Game of Life Output

You may notice some common patterns in the figure. After so many iterations through the game of life rules, only a few cells tend to stay alive. We started with a large random number of alive cells and over time those cells died off. In a controlled environment you may begin with carefully placed live cells and monitor the patterns that emerge to model some other natural phenomena.

Figure: Common Game of Life Surviving and Oscillating Patterns

A New Kind of Science

The name Stephan Wolfram has been mentioned several times in this post. He is the founder of Wolfram|Research, his company is known for the popular Mathematica software suite and Wolfram|Alpha knowledge engine. He did not initially discover cellular automata but recently he has been a prominent figure in its advocacy. He spent 10 years working on his book, A New Kind of Science. In the 1300 page tome, he discusses how cellular automata can be applied to every field of science from biology to physics. NKA is a detailed study of cellular automata programs.

Basic Cellular Automata

Figure: Wolfram's Elementary CA Rule 30. Look at 3 bit input and 1 bit output.

The diagram above depicts the rule 30 program (or rule 30 elementary cellular automaton). There are 8 input states (2 ^ 3) and an output state of one or zero. If you look at the diagram from left to right. The first sequence of blocks on the left depict an input state of { 1 1 1 } with an output of 0. Given input of cells { 1 1 1}, the output will be set to 0. Subsequently, the next set of blocks consist of an input state of { 1 1 0 } with an output of 0.

Here is python pseudo code for processing rule30 input:

```
def rule30(inputCell_0, inputCell_1, inputCell_1): {
if inputCell_0 == 1 and inputCell_1 == 1 and inputCell_2 == 1
return 0:
else if inputCell_0 == 1 and inputCell_1 == 1 and inputCell_2 == 0:
return 0:
...
}
grid = new Grid(100, 100)
grid[row0, col50] = 1 # Enable first cell on row zero
for j until 100:
for i until 100:
valsForNextRow[i] = rule30(inputLastRow[i - 1], inputLastRow[i], inputLastRow[i +
1])



Example of first three cases using a boolean notation:
{ 1 1 1 } -> 0
```

```
{ 1 1 0 } -> 0
{ 1 0 1 } -> 0
...
Example of first few cases with Scala programming language:

class CellularAutomataRule extends Rule {
def rule(inputState:(Int, Int, Int)) : Rules.Output =
inputState match {
case (1, 1, 1) => 0
case (1, 1, 0) => 0
case (1, 0, 1) => 0
case (1, 0, 0) => 1
...
}
} // End of Rule
```

Figure: Scala Example with pattern matching

Figure: Elementary Automata Grid after several iterations, look at image from top to bottom
Cellular Automata and Squaring Application

How do you square two numbers?

With most popular programming languages you could use infix notation providing an input parameter on the left and an input parameter on the right side of some arithmetic function. With Java, you might write the following code:

```
int x = 4 * 4;
Output : 16
```

The above snippet is valid code used to multiply four times four with a result of sixteen but it does not say much about the native implementation of the multiplication operator. There are many layers involved with that particular function but they aren't visible to the developer. Is the function implemented and optimized by the compiler or implemented by the runtime environment? It is possible that the operating system may cache the result or build an implementation for the arithmetic operation. Ultimately for most basic integer multiplication or addition, those operations are performed at the hardware level. So how then does the hardware do it?

In the figure depicted below is an AND gate and truth table, the gate takes two Boolean input values and returns the output AND operation. If one is entered in input A and zero is entered into input B, then the output C returned by the AND gate is one. An arithmetic logic unit may perform basic Boolean operations or possibly some form of basic arithmetic. An ALU may consist of AND, XOR and other similar simple gates combined to ultimately perform basic arithmetic, increment, decrement or jump operations. (Most of my comments focus on older generation basic circuits, modern circuit design may not use such techniques or basic components)

Figure: Boolean AND Gate, InputA, B and Output
If you start from that basic piece of Java code 4 * 4, there are many levels of software and hardware layers that are involved to implement that operation and then return a result.

I wanted to present basic Boolean arithmetic so that you can see how basic rules can lead to

more complex patterns and behavior. One two input AND gate will generate a Boolean result. Several million logic circuits may be used to build a complete CPU. You may already be familiar with the Conway's game of life, an initial grid is created with a random number of initial live cells. We can use a simple cellular automata program to square two integers use the rules described in Wolfram's A New Kind of Science. After so many iterations, a common pattern will emerge and that pattern holds the result of N * N. In our squaring example we started with the input number of enabled cells (N = 4) and after so many iterations a pattern emerged that contained the squaring of the input. In many of Wolfram's Elementary rules, a binary sequence is used for input and output. With the general CA squaring rule, an input and output number ranging from 0 to 7 are defined for each cell.

Squaring Rule

Figure: Applet Visual Output Grid for Squaring Cellular Automata

```
CellularAutomaton[{
{ 0, Blank[], 3} -> 0,
{ Blank[], 2, 3} -> 3,
{ 1, 1, 3 }   -> 4,
{ Blank[], 1, 4} -> 4,
{ Alternatives[1, 2], 3, Blank[]} -> 5,
{ Pattern[$`p, Alternatives[0, 1]], 4, Blank[]} -> 7 - $`p,
{ 7, 2, 6} -> 3,
{ 7, Blank[], Blank[]} -> 7,
{ Blank[], 7, Pattern[$`p, Alternatives[1, 2]]} -> $`p,
{ Blank[], Pattern[$`p, Alternatives[5, 6]], Blank[]} -> 7 - $`p,
{ Alternatives[5, 6], Pattern[$`p, Alternatives[1, 2]], Blank[]} -> 7 - $`p,
{ Alternatives[5, 6], 0, 0} -> 1,
{ Blank[], Pattern[$`p, Alternatives[1, 2]], Blank[]} -> $`p,
{ Blank[],  Blank[], Blank[]} -> 0}, {
...
Append[Table[1, {$CellContext`n$$}], 3], 0},
Table -> Expression to N
Append -> Table to 3
```

Figure: Notebook Source File For Mathematica, General CA Rule for Squaring Automaton

The general rules for the squaring automaton are similar to the rules that were mentioned for the elementary rule30 program. Integer values (range 0 - 7) are used instead of binary inputs and outputs. The initial row and initial number of cells are represented by the input parameter (N = 4 in our example).

```
Example Row: 0 0 0 0 0 3 3 3 3 1 0 0 0 0
```

Besides the first row, the initial grid contains all zeros. On the next sequence, the CA rule for squaring is run against each cell on the second row. On the sequence after that, the CA rule is run against the third row and so on until the last row in the grid has been reached. With a 100 x 100 grid, the output pattern will emerge before row 100 is reached.

```
class SquaringRule extends Rules.GeneralRule {
def ruleId() = 132
def rule(inputState:Rules.RuleInput) : Rules.Output =
inputState match {
case (0, _, 3) => 0
case (_, 2, 3) => 3
```

```
case (1, 1, 3) => 4
case (_, 1, 4) => 4
case (1 | 2, 3, _) => 5
case (0 | 1, 4, _) => 7 - inputState._1
case (7, 2, 6) => 3
case (7, _, _) => 7
case (_, 7, 1 | 2) => inputState._3
case (_, 5 | 6, _) => 7 - inputState._2
case (5 | 6, 1 | 2, _) => 7 - inputState._2
case (5 | 6, 0, 0) => 1
case (_, 1 | 2, _) => inputState._2
case _   => 0
}
} // End of Rule
```

Figure: Scala Source for Squaring Rule uses Pattern Matching

Applied Cellular Automata

Cellular automata is often used with data compression, cryptography, artificial intelligence, urban planning, financial market modeling, music generation, and 3D terrain generation. If you are a software engineer, you may have to step back and consider how cellular automata patterns emerge and understand the nature of the dynamic system before looking for a typical software library. CA is not normally seen in everyday applications. Consider this when you look at some random pattern, don't think of the phenomenon as a random sequence of events that cannot be replicated, think of the event in terms of a cellular automaton. Try to imagine the rules that could model that natural behavior. Modeling seemingly random patterns is an area where cellular automata is being widely used. Urban planning departments are integrating geographic information systems (GIS) with cellular automata in an attempt to predict growth in an area of a city.

Summary

The simple squaring example mentioned in this post merely gives you an overview of a basic cellular automata system. Scientists, biologists, computer scientists and software engineers want to find better ways to observe relationships and patterns that occur in our world. Review Stephen Wolfram's A New Kind of Science to give you an idea for what is possible with seemingly simple rules.

Source Code and Applet

1.
2.
3.
4.
5.

Figure: Squaring Cellular Automaton Output, Input = 4 (top of grid), Output = 16 (pattern towards the bottom)

Resources

1.
2.

--- Berlin Brown (2012)