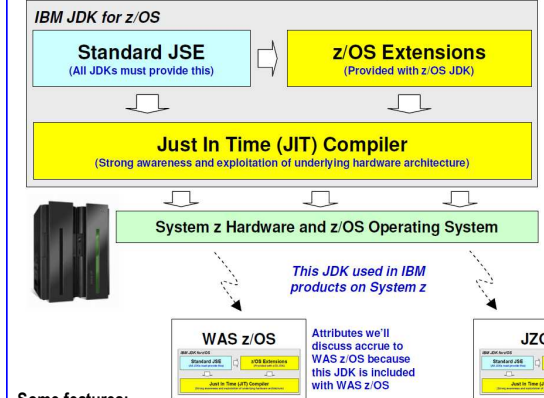


Java is Java, but JDKs are not alike... and not all JDKs are fully aware of and exploit the underlying platform.

- Starting with Java 5, IBM re-wrote the JVM from the ground up employing its extensive knowledge in operating system design and platform-exploiting Just-In-Time compilers.
- A while back IBM set out to completely rewrite its JVM from the ground up and was done in what's called a "clean room" so that no Sun intellectual property is part of the IBM JVM.
- This rewrite also allowed IBM to use its very extensive knowledge of Java and general operating system design techniques in the creation of this new JVM.
- The JVM -- called "J9" -- is designed to be scalable across a wide range of products, from small hand-held and embedded devices all the way up to System z (the name 'J9' has some IBM heritage associated to it).
- All would run the same core JVM with extensible plugins to provide features needed for the platform.



Functionality beyond the standard JSE

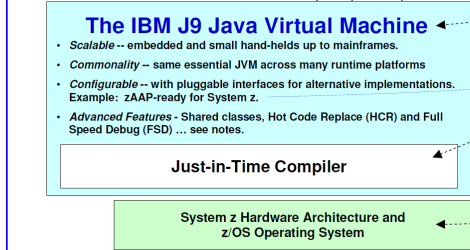
Particularly in the area of security, but a few other things as well as we'll see

Taking advantage of System z hardware

Which manifests itself in efficiency and performance

Some features:

- > **Shared Classes** -- ability for multiple JVM environments to use the same classes held in a shared memory pool.
 - >> Shared libraries are files used by multiple applications where each shared library consists of a symbolic name, a Java class path, and a native path for loading Java Native Interface (JNI) libraries.
 - Example** - Instead of having four copies of my_sample.jar on your system after the four applications are deployed, you can define a shared library for my_sample.jar and have the four deployed applications use that one my_sample.jar library file.
 - With a Shared Library implementation there are caveats due to the way storage is allocated, so plan accordingly. They appear to be carved out of Subpool 230 (High Private). Be aware of new address space storage requirements for those installations migrating to zOS v1.11.
- >> Using shared libraries to reduce the number of duplicate library files on your system, you reduce duplicated memory consumption and it makes for faster JVM startup as classes may simply be fetched from the shared class pool.
- >> Isolated shared libraries provide another way to reduce the number of library files where each have their own class loader, enabling a single instance of the classes to be shared across the applications.
- >> Each application can specify which isolated shared libraries that it wants to reference and can reference different versions of the isolated shared library, resulting in a set of applications sharing an isolated shared library.
- Example** - With isolated shared libraries, some applications can share a single copy of Library A, Version 1 while other applications share a single copy of Library A, Version 2, for a total of two instances in memory.
- >> Using the WAS administrative console, you can define shared libraries for the library files that multiple applications use and then associate the libraries with specific applications or modules or with an application server.
- > **Hot Code Replace (HCR)** -- ability to swap in a new class file and have the JVM swap it with the existing class file.
 - >> HCR has been specifically added as a standard technique to Java to facilitate experimental development and to foster iterative trial-and-error coding.
 - >> HCR can be used to change the body of a method.
 - ! HCR only works when the class signature does not change; you cannot remove or add fields to existing classes.
- > **Full Speed Debug** -- ability to run the JIT in debug mode at full speed allowing you to run your application with all the performance benefits of JIT compiled code without losing the ability to perform some of the most common debugging activities, such as setting breakpoints, stepping through code, and viewing local variables.
 - Use in conjunction with **Hot Code Replace** to do on-the-fly testing and debugging of issues.
 - Within the JVM is the Just In Time compiler (the JIT) that turns Java bytecode into platform readable code.



Decades of IBM operating system and memory management experience

zAAP engines are Java offload engines and a function of the Java SDK and z/OS Dispatcher. They enhance the financial picture of z/OS and free up GPs for other key processes. The zAAP-enabled Java SDK is packaged with WAS z/OS, and takes advantage of zAAPs if they're present and configured. WAS itself knows nothing about this engine switch during runtime.

JIT z/OS development team

System z and z/OS engineering teams

Close Cooperation

CheatSheet #56 zTidBits System z IBM's J9 JVM

IBM 64-bit SDK for z/OS, Java Technology Edition, V6 is the z/OS 64-bit Java product that supplies the Java SDK 6 APIs.

The IBM 64-bit SDK for z/OS, Java Technology Edition, V6 requires z/OS Version 1 Release 7 or higher.

The System z10 processor includes large page support, another key feature for 64-bit performance.

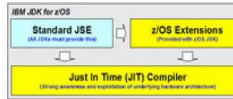
NOTE1: Here we need to make sure a key point is understood -- the JVM on each platform has a unique platform JIT engine. The JIT for z/OS is well aware of the z/OS operating system. In fact, it makes extensive use of it. The developers of the JIT for z/OS worked extensively with the hardware and software engineers for System z and z/OS to make sure the integration was tight and fully exploitive.

NOTE2: Basic message is one of advanced JVM adhering to standards but exploiting platform specifics where possible. The previous illustration below provides another key point about IBM's Java ... while it is fully compliant with the standards defined for a Java Standard Edition (Java SE, or JSE), it goes beyond that.

- Going beyond the standard is perfectly acceptable since IBM extends Java to provide access to key z/OS functions, and the JIT, as explained, makes extensive use of the lower level hardware.
- This is the **J9 JVM** which then gets used in all our IBM products.

NOTE3: Mentioned in the first diagram is **JZOS+** because it provides even more Java-extensions for the mainframe. The **IBM JZOS Batch Toolkit** for z/OS SDKs is a set of tools that addresses many of the functional and environmental shortcomings in current Java batch capabilities on z/OS. It includes a native launcher for running Java applications directly as batch jobs or started tasks, and a set of Java methods that make access to traditional z/OS data and key system services directly available from Java applications.

• The JIT is very much aware of the System z hardware features and exploits them directly for greater throughput and efficiency evolving over the years to have some very sophisticated underlying features.



JIT in JDK for z/OS is written to exploit :

- **Superscalar Dual-Pipeline**
 - The z890, z990, z9 and z10 are superscalar dual pipeline designs.
 - > It permits the dispatching of three instructions per cycle.
 - > The JIT understand this generating code that exploits this where possible.
- **Register Allocation**
 - > 13 of the 16 64-bit processor registers are available and the JIT makes extensive and efficient use of the registers to enhance throughput and performance.
- **Compare-and-Swap**
 - > Compare-and-Swap is a feature that allows programs to use comparison of register values to provide a form of code access lock management.
 - > It's very efficient where the JIT uses this for Java synchronization.
- **Private Linkage**
 - > Program linkage at the lowest possible level, allowing one method to call another with a minimum of overhead. Includes direct JNI dispatch from JIT.
- **CISC Exploitation**
 - > System z hardware is a "Complex Instruction Set Computer" (CISC) design, with many very elaborate instruction features.
 - > The JIT knows about these and exploits them; for example, Java loop structures reduced to a very small set of CISC instructions.
- **64-Bit Instructions and 31-Bit JVMs**
 - > The System z hardware is a 64-bit design.
 - > If the JVM is running in 31-bit mode the JIT is capable of exploiting the 64-bit hardware for long arithmetic operations.
- **Gryphon extensions**
 - > The new mainframe likely will have hybrid enhancements to exploit features increasing performance and throughput.



In addition to the standard compliant Java there are extensions to provide exploitations of System z and z/OS functions.

• The IBM 64-bit SDK for z/OS, V6 and V7 provides a full function SDK compliant with the SDK APIs. Extensions to the JSE

Documentation is available for content that is additional to the base.

- Security functions:
 - o Java Cryptography Extension (IBMJCE)
 - o Java Cryptography Extension in Java 2 Platform Standard Edition, Hardware Cryptography (IBMJCECCA)
 - o Java Secure Sockets Extension (IBMJSSE)
 - o Java Certification Path (CertPath)
 - o Java Authentication and Authorization Service (JAAS)
 - o SAF interfaces
 - o Java Generic Security Services (JGSS)
 - o Java PKCS#11 Implementation Provider (IBMPKCS11Impl)
- RMI-IIOP
- Java Record I/O (JRIO)
- JZOS - Java Batch Launcher and Toolkit

security standards to provide access to System z and z/OS facilities such as SAF and the Crypto Hardware

Access to VSAM files, sequential files, PDS directories and the system catalog

More on JZOS

All content above is shipped with the z/OS SDK product and is zAAP eligible.

See: www.ibm.com/servers/eserver/zseries/software/java/products/j9pcont64.html

- The URL above points you to the Java 6 64-bit page ... change the 64 to 31 and you'll get the 31-bit page containing the same essential information in either case.
- There is a set of links on that page that offer information on extensions to the J9 for z/OS JVM to make use of System z specifics, many of which center around the topic of security ... the exploitation of SAF and the Crypto facilities of the System z platform and is in addition to the standard JSE.
- Applications do not need to be written to this, but they may if the functionality is desired.

† See #34 zTidbits (Java Batch)