Java 9

By CHHAVNISH MITTAL

About Me

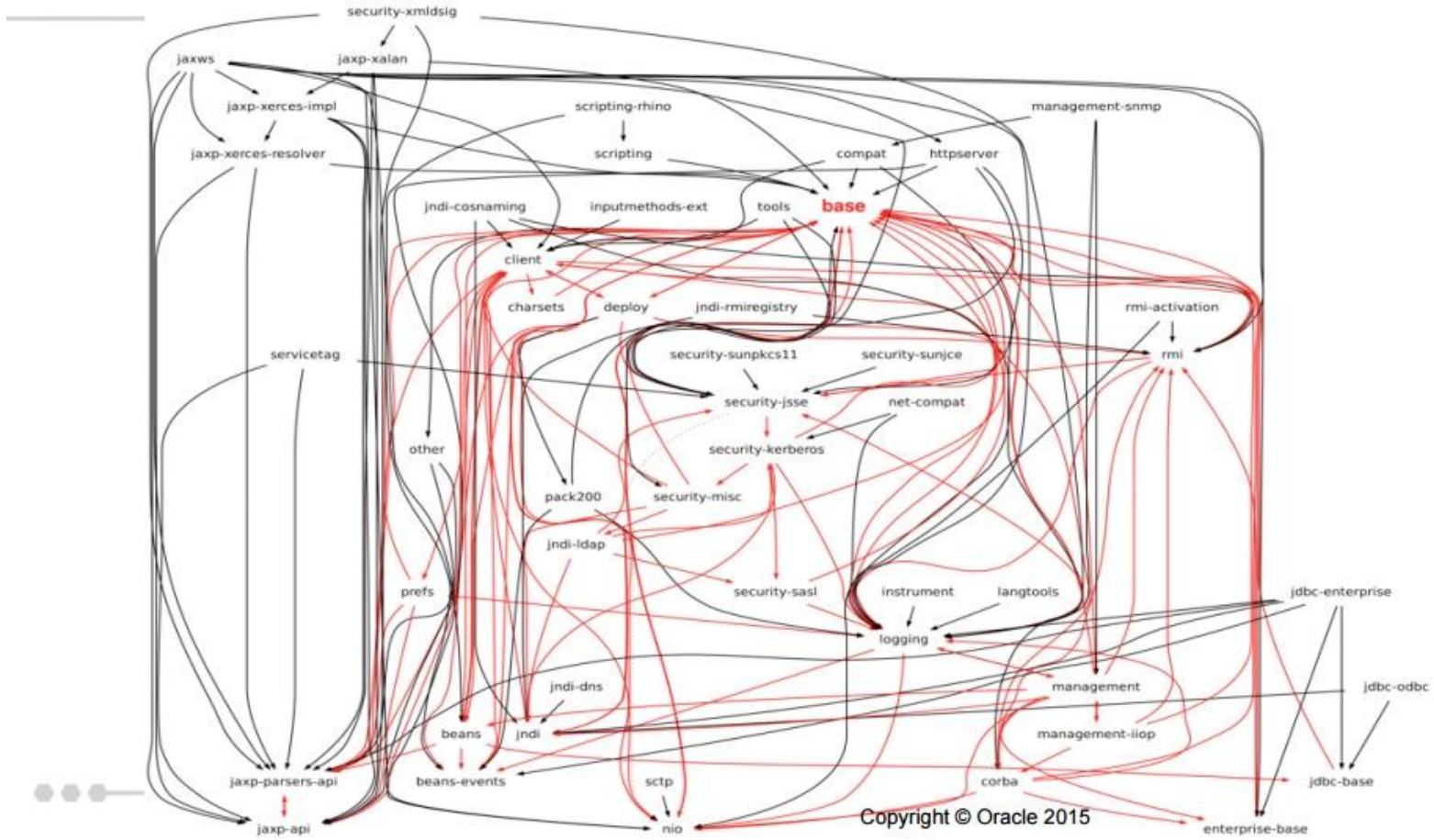
@ChhMittal

- Software Developer at Cellos Software Limited
- Co-Founder and Leader at Delhi & NCR Java User Group
- Author of 2 Research Papers
- Active Member of JCP (Java Community Processes) and OpenJDK
- Grand Finalist of Accenture Innovation Jockey all over INDIA

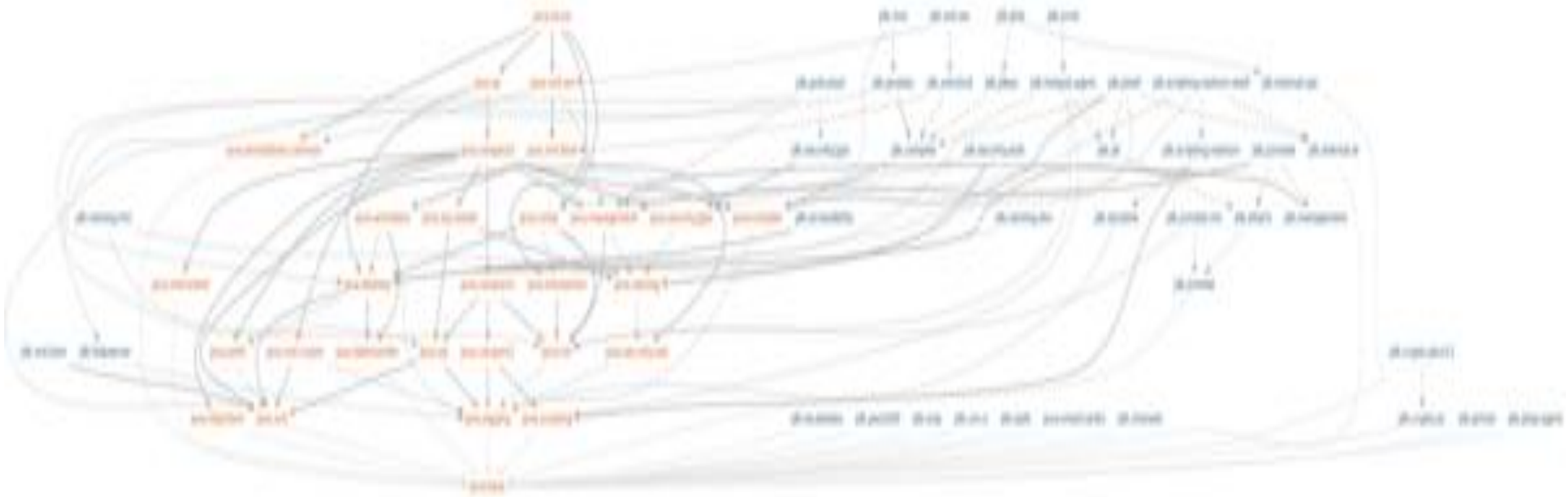
Chhavnish@gmail.com

+91-8826771419

JEP 200


security-xmldsig
jaxws
jaxp-xalan
jaxp-xerces-impl
scripting-rhino
management-snmp
jaxp-xerces-resolver
scripting
compat
httpserver
jndi-cosnaming
inputmethods-ext
tools
base
client
charsets
deploy
jndi-rmiregistry
rmi-activation
servicetag
security-sunpkcs11
security-sunjce
rmi
security-jsse
net-compat
other
security-kerberos
pack200
security-misc
jndi-ldap
prefs
security-sasl
instrument
langtools
jdbc-enterprise
logging
jndi-dns
management
jdbc-odbc
beans
jndi
management-iiop
jaxp-parsers-api
beans-events
sctp
corba
jdbc-base
jaxp-api
nio
enterprise-base
Copyright © Oracle 2015

JEP 200



JEP 200

- More Easily Scalable down to Small Devices
- More Security and Maintainability
- Improved Application Performance

JEP 201: Modular Source Code

- Most of the JDK source code is today organized, roughly, in a scheme that dates back to 1997. In abbreviated form:
  src/{share,$OS}/{classes,native}/$PACKAGE/*.{java,c,h,cpp,hpp}
- The modularization of the JDK presents a rare opportunity to completely restructure the source code in order to make it easier to maintain. We propose to implement the following scheme in every repository in the JDK forest except for hotspot. In abbreviated form:
  src/$MODULE/{share,$OS}/classes/$PACKAGE/*.java
  native/include/*.{h,hpp}
  $LIBRARY/*.{c,cpp}
  conf/*

JEP 220: Modular Run Time Images

- Moving towards jmods from jar which was based on ancient zip format
- Locate and load class and resource files on a per-module basis
- Selective *de-privileging* of JDK classes that today are granted all security permissions but do not actually require those permissions
- Rt.jar and tools.jar removed and are converted to 95 jmods

JEP 222: jshell -> (Read-Eval-Print, Loop)

- No need of creating whole class and writing public static void main….just to test single or group of statements
- It is just for testing syntax of single/multiple line of code only
- Mainly helpful for learning purpose specially in college

JEP 282: jlink The Java Linker

- Optional phase between compile time and run time
- Create your own runtime image with just the modules you need resulting in smaller image
- Very useful for small devices which have very less space
- Creating own image for docker which will consume less memory
- $ jlink --module-path <modulepath> --add-modules <modules> --limit-modules <modules> --output <path>
- Any transitive dependency must be resolved

JEP 269: Factory methods for Collections

- New Convivence methods for List, Sets and Maps interfaces
- Useful when adding multiple elements while creating a unmodifiable Set, List or Map
- Set.of(a,b,c,d); Map.of(a,b,c,d);

JEP 238: Multi Release Jar Files

- Classes of different Java Versions will be invoked depending upon the java image installed on that platform
- Suppose application is designed according to java 8 but later decided to use features of java 9 and java 10 as well then directory structure of our application will be designed as:
- It will first search for platform specific class then to root directory

```
jar root
 - A.class
 - B.class
 - C.class
 - D.class
 - META-INF
   - versions
     - 9
       - A.class
       - B.class
     - 10
       - A.class
```

JEP 266: Concurrency Updates

- Reactive Streams, Publish-Subscriber Framework
- A utility class SubmissionPublisher is provided that developers can use to create custom components.
- CompletableFuture API Enhancements
  - delayedExecutor allow a task to execute after a certain duration
  - Subclass enhancements are added making it easier to extend from CompletableFuture, such as to provide a subclass that supports an alternative default executor

JEP 277: Enhance Deprecation

- Lot of confusion among developers as only few Deprecated marked API were actually removed.
- forRemoval() method returning Boolean to check weather particular API element will be removed in future or not.
- A method named since() returning String. This string should contain the release or version number at which this API became deprecated.
- static analysis tool jdeprscan will be provided that scans a jar file for uses of deprecated API elements.

JEP 213: Milling Project Coin

- Single underscore (_) is not a valid identifier anymore and will be used for lambdas
- Private methods are supported in interfaces
- Effectively final variables in try-with-resources
- Allow @SafeVargs on private instance methods. Since the only constraint on @SafeVargs was that methods should not be overridden. Therefore, it would be reasonable to allow @SafeVarargs to be applied to private methods too.
- Diamond Operator in anonymous class.

JEP 227/267: Unicode 7.0/8.0 Supported

- Two new Unicode Versions were released since Java 8
- 2,834 new characters are included in Unicode7.0
- 7,716 new characters are included in Unicode8.0

JEP 229: Create PKCS12 Keystrokes by Default

- Changing default keystroke type from JKS to PKCS12
- Improved Security
- Option to choose between type of keystroke you want to use

JEP 224: HTML Javadoc

- A command-line option is added to the standard doclet to request a specific type of output markup. HTML4, which is the current type, will be the default. HTML5 will become the default in JDK 10.
- The semantic value of the generated HTML is improved by using structural HTML5 elements such as header, footer, nav, etc.
- The -Xdoclint feature is update to check documentation comments for common errors, based upon the requested type of output markup.

JEP 287: SHA3 Hash Algorithm

- Keeping Java Updated with new SHA3 Algorithm
- The previous SHA 2 Algorithm was published 10 years ago and therefore Hackers have ways to crack it.

JEP 288: Disable SHA-1 Certificates

- Only X.509 certificate chains that are validated by the PKIX implementation of the CertPathValidator and CertPathBuilder APIs and the SunX509 and PKIX implementations of the TrustManagerFactory API are subject to the restrictions.
- A new constraint named jdkCA, that when set, restricts the algorithm if it is used in a certificate chain that is anchored by a trust anchor that is pre-installed in the JDK cacerts keystore.
- A new constraint named denyAfter, that when set, restricts the algorithm if it is used in a certificate chain after the specified date.
- A new constraint named usage, that when set, restricts the algorithm if it is used in a certificate chain for the specified use(s). Three usages are initially supported: TLSServer for TLS/SSL server certificate chains, TLSClient for TLS/SSL client certificate chains, and SignedJAR for certificate chains used with signed JARs.

JEP 226: UTF-8 Property Resource Bundles

- Now use UTF-8 for encoding property files
- ResourceBundle API is extended to Load them
- No need to convert property file using escape mechanism

JEP 273: DRGB-based Secured Random Impl.

- Deterministic Random Bit Generator Implemented
- New methods for SecureRandom matching 800-90C, which allows configuration of a SecureRandom object and specifying additional input in the course of seeding, reseeding, and random-bit generation.
- A new SecureRandomParameters interface so that additional input can be provided to the new SecureRandom methods.
- Implement the three DRBG mechanisms (Hash_DRBG, HMAC_DRBG, CTR_DRBG) in 800-90Ar1 (on all platforms).

JEP 268: XML Catalogs

- The CatalogManager will manage the creation of XML Catalogs and CatalogResolvers, as well as features and properties.
- A Catalog will implement the semantics of OASIS Open Catalog files. It will define an entity catalog that maps external identifiers and URI references to (other) URI references, and delegates to other catalogs.
- A CatalogResolver will implement JAXP's existing EntityResolver and URIResolver interfaces. The resolver will supports the OASIS standard processing instruction as a SAX XMLFilter.

JEP 248: Make G1 Default Garbage Collector

- Advantage of low-pause collector then current throughput oriented collector Parallel GC.
- G1 is now quite mature due to several enhancements during Java 8 release
- Pause time is proportional to heap size.

JEP 254: Compact Strings

- Save a lot of Space
- Migration from 2 bytes per character to 1 byte/2 byte depending upon the characters of the string

JEP 250: Store Interned Strings in CDS Archives

- Reduce memory consumption by sharing the String objects and underlying char array objects amongst different JVM processes.
- Only Supported for G1 Garbage Collector
- Not Supported for 32 bit machines

JEP 265: Marlin Graphics Renderer

- Current graphic rasterizer Pisces is crucial for many graphic intensive applications but Marlin Graphic rasterizer will perform better then current in some cases
- Multi threaded Scalability will outperform performance then Pisces
- The Goal is either equal or higher performance(in some situations) then Pisces

JEP 143: Improved Contended Locking

- No impact on our coding process.
- There will be a considerable improvement in performance of the following:
  - Field reordering and cache line alignment
  - Speed up PlatformEvent::unpark()
  - Fast Java monitor enter operations
  - Fast Java monitor exit operations
  - Fast Java monitor notify/notifyAll operations

JEP 246: Leverage CPU Instructions for GHASH and RSA

- GHASH, which is part of GCM, will be accelerated using pclmulqdq on Intel x64 and xmul/xmulhi on SPARC.
- RSA will be accelerated by using Bit Manipulation Instruction Set 2. It is likely that other asymmetric algorithms will benefit from from these changes, but they will be measured by RSA.
- A new security property has been added to the java.security file, jdk.security.provider.preferred, to allow certain algorithm and algorithm groups to be directed to a particular provider before the ordered provider list is checked.
- 34-150x better performance for certain tests

JEP 257: Update JavaFX/Media to new version of GStreamer

- Updated new version of GStreamer to 1.4.4 released in 2014
- Made required changes to plugins to solve compatibility issues.
- Updated Glib to latest version which was essential for updating Gstreamer.

JEP 197: Segmented Code cache

- Divide the code cache into distinct segments, each of which contains compiled code of a particular type, in order to improve performance and enable future extensions
- Separate non-method, profiled, and non-profiled code
- The following command-line switches are introduced to control the sizes of the code heaps:
- -XX:NonProfiledCodeHeapSize: Sets the size in bytes of the code heap containing non-profiled methods.
- -XX:ProfiledCodeHeapSize: Sets the size in bytes of the code heap containing profiled methods.
- -XX:NonMethodCodeHeapSize: Sets the size in bytes of the code heap containing non-method code.

JEP 158: Unified JVM Logging

- Common command line options for all logging
- Logging is performed at different levels: error, warning, info, debug, trace, develop
- Log messages are categorized using tags (e.g. compiler, gc, classload, metaspace, svc, jfr, ...). One message can have multiple tags (tag-set)
- Logging messages are decorated with information about the message.

JEP 271: Unified GC Logging

- Re-implement GC logging using JVM logging introduced in JEP 158
- Many Command line options have been changed

JEP 285: Spin-Wait Hints

- Allow java code to hint the run-time system that it is in spin loop
- New method onSpinWait();
- The reaction time of a spin loop may be improved when a spin hint is used due to various factors, reducing thread-to-thread latencies in spinning wait situations
- The power consumed by the core or hardware thread involved in the spin loop may be reduced, benefiting the overall power consumption of a program, and possibly allowing other cores or hardware threads to execute at faster speeds within the same power consumption envelope.

JEP 193: Variable Handles

- Partial Replacement of sun.misc.unsafe
- Fence Operations
  - Fine grained Memory Control
  - Atomic operations on Object Fields and array elements
- VarHandle
  - compareAndExchange(), compareAndSet()
  - getAndAdd(), getAndSet()
  - aquireFence(), releaseFence()

JEP 274: Enhanced Method Handles

- Mostly Used by people who develop languages not application
- Support for loops and try-finally blocks which was not provided by MethodHandles
- Better argument Handling with following:
  - Spreading
  - Collection
  - Folding
- Lookup API should allow for looking up classes from different contexts, which is currently not possible

JEP 165: Compiler Control

- Control of C1/C2 JIT, not javac
- Directives option List:
  - Common flags: Enable, bool Exclude, bool BreakAtExecute, bool BreakAtCompile, bool Log, bool PrintAssembly, bool PrintInlining, bool PrintNMethods, bool ReplayInline, bool DumpReplay, bool DumpInline, bool CompilerDirectivesIgnoreCompileCommands, bool Inline, ccstr[]
  - C2 only: BlockLayoutByFrequency, bool PrintOptoAssembly, bool PrintIntrinsics, bool raceOptoPipelining, bool TraceOptoOutput, bool TraceSpilling, bool Vectorize, bool VectorizeDebug, bool CloneMapDebug, bool IGVPrintLevel, intx MaxNodeLimit, intx DisableIntrinsics, ccstr
- Runtime changes via jcmd

JEP 102: Process API Updates

- The java.lang.Process class is enhanced to provide the operating specific process id of the process, information about the process including the arguments, the command, the start time of the process, the accumulated cpu time of the process and the user name for the process.
- The java.lang.ProcessHandle class returns information about each process as provided by the operating system including process id, arguments, command, start time, etc. A ProcessHandle can return the process' parent, and the direct children, and to all descendants via a stream of ProcessHandles.
- ProcessHandles can be used to destroy processes and monitor process liveness. With ProcessHandle.onExit, the asynchronous mechanisms of CompletableFuture can be used to schedule an action to be taken when the process exits.

JEP 223: New Version-String Scheme

- JDK $MAJOR.$MINOR.$SECURITY.$PATCH
- More human readalble and understandable
- Semantic versioning

JEP 225: Javadoc Search

- Search Box Added to API Documentation generated by standard doclet
- Declared names of modules, packages, types and members are indexed and searchable. Since methods can be overloaded, the simple name of method parameter types are also indexed and can be searched for. The method parameter names are not indexed.
- A search term or a phrase indexed using a new inline tag, @index, are searchable. Other inline tags cannot be nested inside @index.

JEP 217: Annotations Pipeline 2.0

- Redesign the javac annotations pipeline to better address the requirements of annotations and tools that process annotations
- Repeating type and lambda annotation in JDK 8

JEP 236: Parser API for NASHORN

- Interface classes to represent Nashorn syntax-tree.
- Factory to create a configured parser instance, with configuration done by passing Nashorn command-line options via an API.
- Visitor-pattern API to visit AST nodes.
- The parser API will enable ECMAScript code analysis by programs such as IDEs and server-side frameworks without those programs having to depend on Nashorn's internal implementation classes.

JEP 289: Deprecate the Applet API

- Not many people use it
- Many browsers doesn't support it

JEP 231: Remove Launch time JRE version selection

- JRE version selection no longer works at run time
- However you can still see the version you are using of JRE

JEP 298: Remove Demos and Samples

- Following demos and samples were removed as they were out of date and unmaintained
  - demo/share/jvmti
  - demo/share/management
  - demo/share/nbproject
  - demo/share/scripting
  - demo/solaris/jni
  - sample/share/annotations
  - sample/share/forkjoin
  - sample/share/jmx
  - sample/share/lambda
  - sample/share/nio
  - sample/share/scripting
  - sample/share/try-with-resources
  - sample/share/vm
  - sample/solaris/dtrace

JEP 240: Remove the JVM TI hprof Agent

- The useful features of the hprof agent have been superseded by better alternatives.
- Heap dumps
  - This functionality has been superseded by the same functionality in the JVM. Using the Diagnostic Command GC.heap_dump (jcmd <pid> GC.heap_dump)
- Allocation profiler
  - The Java VisualVM tool provides the same functionality, as do a number of third-party profilers.
- CPU profiler
  - The CPU profiler has a couple of shortcomings. Java VisualVM and Java Flight Recorder, which are bundled with the JDK, as well as a number of third-party profilers provide better functionality.

JEP 241: Remove of jhat tool

- Jhat was added in jdk6 and was for experimental purpose only
- Only few people uses it so impact is very less

JEP 214: Remove GC Combinations deprecated in JDK 8

- Following Flags and Flag Combinations will stop working
  - DefNew + CMS : -XX:-UseParNewGC -XX:+UseConcMarkSweepGC
  - ParNew + SerialOld : -XX:+UseParNewGC
  - ParNew + iCMS : -Xincgc
  - ParNew + iCMS : -XX:+CMSIncrementalMode -XX:+UseConcMarkSweepGC
  - DefNew + iCMS : -XX:+CMSIncrementalMode -XX:+UseConcMarkSweepGC -XX:-UseParNewGC
  - CMS foreground : -XX:+UseCMSCompactAtFullCollection
  - CMS foreground : -XX:+CMSFullGCsBeforeCompaction
  - CMS foreground : -XX:+UseCMSCollectionPassing

JEP 110: HTTP2 Client

- New HTTP client API that implements HTTP/2 and WebSocket that replace the legacy HttpURLConnection API
- Easy to use
- Notification of events such as "headers received", errors, and "response body received“
- Friendly towards new language features such as lambda expressions
- The prototype API also included:
  - Separate requests and responses, like the Servlet and HTTP server API;
  - Asynchronous notification of the following events:
    - Response headers received,
    - Response error,
    - Response body received, and
    - Server push (HTTP/2 only);
  - HTTPS, via SSLEngine;
  - Proxying;
  - Cookies; and
  - Authentication.

JEP 211: Elide Deprecation warnings on import statements

- Making large code bases clean of lint warnings
- The deprecation warnings on imports cannot be suppressed using the @SuppressWarnings annotation, unlike uses of deprecated members in code which will be removed

JEP 215: Tiered Attribution for javac

- Improved performances by implementing an approach which reduces the number of (redundant) passes needed to attribute a given expression
- Same results as the current type-checking implementation
- New structural types will be created for the following argument expressions:
  - Lambda expressions,
  - Conditional poly expressions,
  - Generic method calls,
  - Parenthesized poly expressions,
  - Method references, and
  - Diamond instance creation expressions.

JEP 219: Datagram Transport Layer Security

- Protocols that support DTLS include, but are not limited, to:
  - RFC 5238, Datagram Transport Layer Security (DTLS) over the Datagram Congestion Control Protocol (DCCP)
  - RFC 6083, Datagram Transport Layer Security (DTLS) for Stream Control Transmission Protocol (SCTP)
  - RFC 5764, Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)
  - RFC 7252, Constrained Application Protocol (CoAP)
- Google Chrome and Firefox now support DTLS-SRTP for Web Real-Time Communication (WebRTC).

JEP 221: New Doclet API

- Reduce the maintenance burden of outdated APIs.
- Eliminate the use of a custom language-model API in favor of the standard language-model API, javax.lang.model, introduced in Java SE 6.
- Eliminate the simplistic support for analyzing documentation comments in favor of the Compiler Tree API, com.sun.source.doctree, introduced in JDK 8.
- Replace the use of the "template class" com.sun.javadoc.Doclet with a suitable new interface type.

JEP 228: Add more Diagnostic Commands

- print_class_summary
  - Print a list of all loaded classes and their inheritence structure.
  - Responsible group: runtime
- print_codegenlist
  - Print methods queued for compilation with either C1 or C2 (separate queues)
  - Responsible group: compiler
- print_utf8pool
  - Print all UTF-8 string constants.
  - Responsible group: runtime
- datadump_request
  - Signal the JVM to do a data-dump request for JVMTI.
  - Responsible group: serviceability

JEP 228: Add more Diagnostic Commands (Cont….)

- dump_codelist
  - Print n-methods (compiled) with full signature, address range, and state (alive, nonentrant, and zombie).
  - Allow selection of printing to stdout or to a file.
  - Allow XML or text printout.
  - Responsible group: compiler
- print_codeblocks
  - Print the size of the code cache and a list of blocks in the code cache, with adresses.
  - Responsible group: compiler
- set_vmflag
  - Set a command-line flag/option in VM or the libraries.
  - Responsible group: servicability

JEP 232: Improved Secure Application Performance

- Use ConcurrentHashMap to map ProtectionDomain to PermissionCollection
- SecureClassLoader should use a ConcurrentHashMap
- Remove Policy provider code that synchronizes on identityPolicyEntries List
- Store PermissionCollection entries in a ConcurrentHashMap instead of a HashMap in Permissions class
- Store permissions in concurrent collections in PermissionCollection subclasses

JEP 233: Generate Run Time Compiler Test Automatically

- The tool will work automatically, without human interaction.
- Compiling source code during test execution is problematic for embedded platforms, where a full JDK might not be available, so the tool will provide a way to pre-compile source-code tests.
- The generated test cases will include complicated expressions and control-flow graphs and will make use of intrinsics, floating-point operations, try-catch-finally constructs, etc

JEP 243: Java Level JVM Compiler Interface

- Dynamic Compiler to be used by JVM
- Allow a Java component programmed against the JVMCI to be loaded at runtime and used by the JVM's compile broker.
- Allow a Java component programmed against the JVMCI to be loaded at runtime and used by trusted Java code to install machine code in the JVM that can be called via a Java reference to the installed code.
- Experimental

JEP 247: Compile for Older Platform Versions

- New Command Line option –release
- automatically configures the compiler to produce class files that will link against an implementation of the given platform version. --release N is roughly equivalent to:
  - for N < 9: -source N -target N -bootclasspath <documented-APIs-from-N>,
  - for N >= 9: -source N -target N --system <documented-APIs-from-N>.

JEP 251: Multi Resolution Images

- The new API, to be defined in the java.awt.image package, will allow a set of images with different resolutions to be encapsulated into a single multi-resolution image.
- The basic operations on a multi-resolution image are:
  - Retrieve a resolution-specific image variant based on a given DPI metric and set of image transformations, and
  - Retrieve all of the variants in the image.

JEP 252: Use CLDR Locale Data by Default

- The Unicode Consortium's Common Locale Data Repository is the de-facto standard for locale data on many platforms. Although CLDR locale data is bundled with the JRE as of JDK 8, it is not enabled by default. To turn it on, the end user must explicitly set the system property java.locale.providers, e.g.:
  - java.locale.providers=JRE,CLDR
- By enabling the use of CLDR data by default, de-facto standard locale data will be provided to users without any further action on their part.

JEP 253: Prepare JavaFX UI Controls & CSS APIs for Modularization

- Define public APIs for the JavaFX UI controls and CSS functionality that is presently only available via internal APIs and will hence become inaccessible due to modularization.
- Make UI control skins into public APIs

JEP 255: Merge Selected Xerces 2.11.0 Updates into JAXP

- Update the JDK with changes in the following categories from Xerces 2.11.0:
  - Datatypes,
  - DOM L3 Serializer,
  - XPointer,
  - Catalog Resolver, and
  - XML Schema Validation (including bug fixes, but not the XML Schema 1.1 development code).
- There will be no changes to the JAXP public API.

JEP 258: HarfBuzz Font-Layout Engine

- The ICU OpenType layout engine is no longer actively developed and IBM
- Incorporate into the JDK a copy of the HarfBuzz library in the same manner as is done for ICU and other open-source libraries in the JDK such as libpng.

JEP 262: TIFF Image I/O

- Suitable TIFF reader and writer plugins, written entirely in Java, were previously developed in the Java Advanced Imaging API Tools Project (javadoc)
- The package will be renamed to javax.imageio.plugins.tiff, since it will become a standard part of the Java SE specification.
- The XML metadata format name will be similarly renamed.

JEP 263: HiDPI Graphics on Windows and Linux

- HiDPI on Windows
- HiDPI on Linux

JEP 264: Platform Logging API and Service

- Defined and used in the java.base module, hence it cannot depend upon the java.util.logging API.
- Be easily adoptable by applications which use external logging framework, such as SLF4J or Log4J.
- Reduce dependences on the java.logging module in order to simplify the JDK module graph.
- Deals with bootstrap issues, so that platform classes can log messages before the log consumer is initialized.
- By default, log messages via the java.util.logging API when the java.logging module is present

JEP 270: Reserved Stack Areas for Critical Sections

- Reserve extra space on thread stacks for use by critical sections, so that they can complete even when stack overflows occur.
- Provide a mechanism to mitigate the risk of deadlocks caused by the corruption of critical data such as java.util.concurrent locks (such as ReentrantLock) caused by a StackOverflowError being thrown in a critical section.
- The solution must be mostly JVM-based in order not to require modifications to java.util.concurrent algorithms or published interfaces, or existing library and application code.
- The solution must not be limited to the ReentrantLock case, and should be applicable to any critical section in privileged code.
- This annotation, currently named jdk.internal.vm.annotation.ReservedStackAccess, is a runtime method annotation that can be used by any class of privileged code

JEP 272: Platform-Specific Desktop Features

- API will be designed to be cross-platform so that each feature can be implemented on as wide a range of platforms as possible.
- apple.applescript classes are being removed without providing any replacement.
- Provide public API to replace the functionality in com.apple.{eawt,eio}
- Provide access to similar features on other platforms

JEP 279: Improve Test-Failure Troubleshooting

- Gather the following information to help diagnose test failures and timeouts:
  - For Java processes which are still running on a host after test failure or timeout:
    - C and Java stacks
    - Core dumps (minidumps on Windows)
    - Heap statistics
  - Environment information:
    - Running processes
    - CPU and I/O loads
    - Open files and sockets
    - Free disk space and memory
    - Most recent system messages and events

JEP 283: Enable GTK 3 on Linux

- Java graphics should be able to support both GTK 2 and GTK 3.
- Major subtasks for AWT/Swing:
  - Adapt the existing dynamic GTK shim to support GTK 3 as well.
  - Rework Swing GTK LnF to support GTK-3.
  - Migrate the FileChooserDialog peer to GTK-3.
  - Migrate AwtRobot to GTK-3 (or maybe remove dependency on any GTK).
  - Test via existing automated tests, both GTK 2 and GTK 3.
- Subtasks for JavaFX:
  - Create a shim to dynamically load GTK 2.
  - Adapt the dynamic GTK shim to support GTK 3 as well.
  - Test via existing automated tests, both GTK2 and GTK 3.

JEP 284: New HotSpot Build System

- Leverage the functionality present in the build-infra framework, to minimize code duplication.
- Simplify the HotSpot build system to provide a more maintainable code base and to lower the threshold for future improvement.

JEP 291: Deprecate the Concurrent Mark Sweep (CMS) Garbage Collector

- Dropping support for CMS and then removing the CMS code, or at least more thoroughly segregating it, will reduce the maintenance burden of the GC code base and accelerate new development. The G1 garbage collector is intended, in the long term, to be a replacement for most uses of CMS.
- warning message is issued when it is requested on the command line, via the -XX:+UseConcMarkSweepGC option.

JEP 292: Implement Selected ECMAScript 6 Features in Nashorn

- ECMAScript 6 includes the following new features:
  - Arrow functions: a concise way of defining functions using => syntax
  - Classes: a way to define classes using inheritance, constructors, and methods
  - Enhanced object literals: support for special and computed property keys
  - Template strings: dynamically-evaluated multiline strings
  - Destructuring assignment: assignment binding using object or array syntax
  - Default, rest, and spread parameters: more flexible argument passing
  - let, const, and block scope: block-scoped declaration of variables and constants
  - Iterators and for..of loops: a protocol to iterate over arbitrary objects
  - Generators: a special kind of function to create iterators
  - Unicode: full Unicode support with backwards compatiblilty
  - Modules: language-level support for the definition of modules
  - Module loaders: support for dynamic loading, isolation, and compilation hooks
  - Map, Set, WeakMap, and WeakSet: various new collection classes
  - Proxies: allows creation of objects with special behaviour
  - Symbols: a new kind of unique property key
  - Subclassable built-ins: built-ins such as Array and Date can be subclassed
  - Promises: an API for asynchronous future completion
  - Math, Number, String, and Object APIs: various new functions on built-in objects
  - Binary and octal literals: new forms for numeric literals
  - Reflection API: API for performing meta-programming operations
  - Tail calls: allow recursive code without unbounded stack growth

JEP 295: Ahead-of-Time Compilation

- Improve the start-up time of both small and large Java applications, with at most a limited impact on peak performance.
- Change the end user's work flow as little as possible.
- To use the AOTed java.base module, the user will have to compile the module and copy the resulting AOT library into the JDK installation directory, or specify it on java command line
- Current AOT limitations
  - AOT initial release in JDK 9 is provided for experimental-only use and is restricted to Linux x64 systems running 64-bit Java with either Parallel or G1 GC.
  - The system must have installed libelf to allow generation of AOT shared libraries (.so) as result of AOT compilation.
  - AOT compilation must be executed on the same system or a system with the same configuration on which AOT code will be used by Java application.
  - The same Java run-time configuration must be used during AOT compilation and execution. For example, the jaotc tool should be run with Parallel GC (using the -J flag) if an application will also use Parallel GC with AOT code. Mismatching run-time configuration may cause application crash during execution.
  - May not compile java code which uses dynamically generated classes and bytecode (lambda expressions, invoke dynamic).

JEP 199: Smart Java Compilation, Phase Two

- No new Features
- Fix sjavac on all platforms in JPRT
- Properly clean up sjavac heuristics
- Refactor the protocol, possibly by leveraging RMI
- Make the option names and syntax more consistent with those of javac
- Replace System.{out,err} logging with something more suitable
- Increase parallelism by sharing more resources in javac, where possible

JEP 216: Process Import Statements Correctly

- In some cases javac will accept source code with a certain order of imports and reject the same source code with just the imports reordered

Thank you