Java SE 9: Continuing to Thrive in the Cloud!

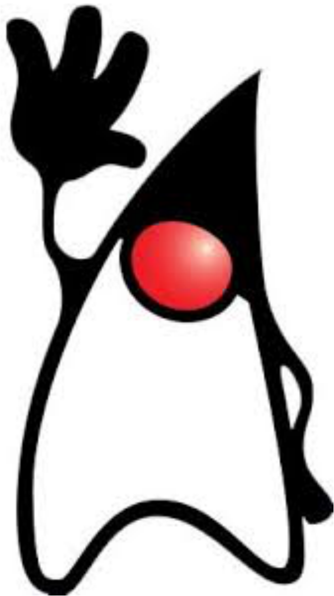
Bernard Traversat
Vice President of Engineering
Java SE Platform, Oracle

Nov 8th, 2016

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Who am I?



- VP of Engineering at Oracle - managing the Java SE development team
  – Everything delivered as part of the JDK/JRE
  – Everything developed in OpenJDK
- Previously at Sun, started to work on Java in 1998 on the JavaOS project
- Before, I worked at NASA Ames on operating systems for massively parallel supercomputers

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

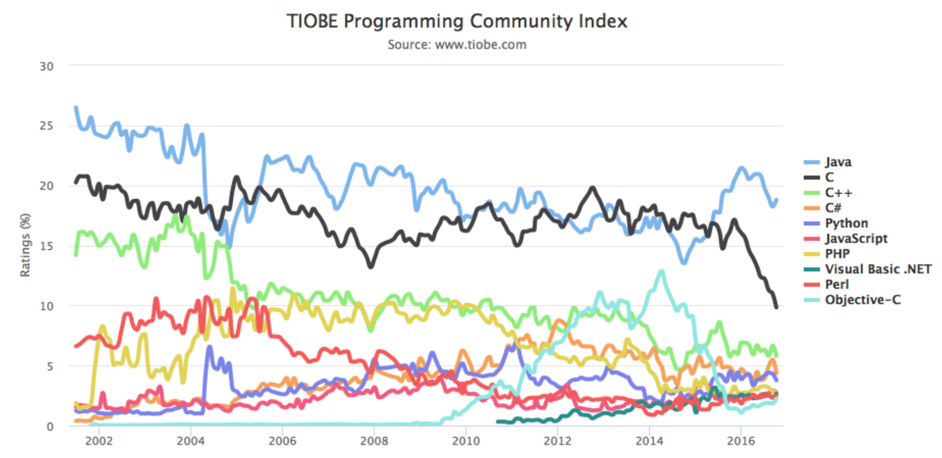
Agenda

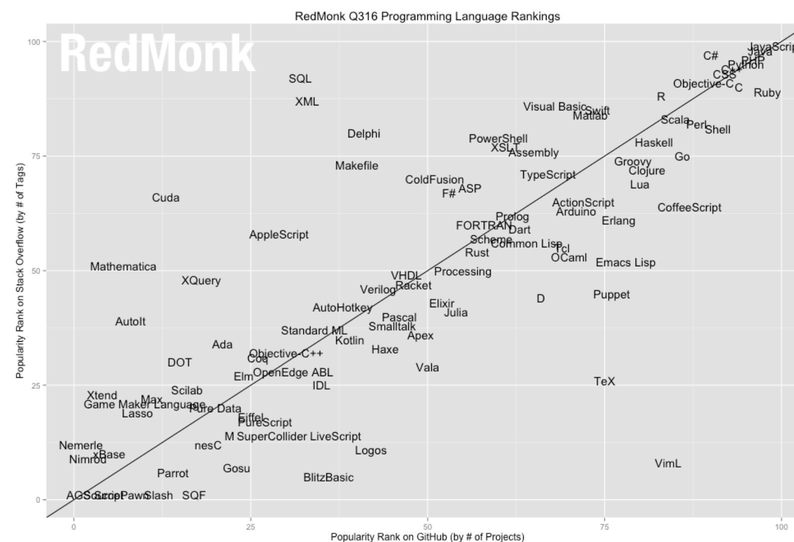
1. Java Adoption
2. Java in the Cloud
3. Java SE 9

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Java SE is #1 with Developers

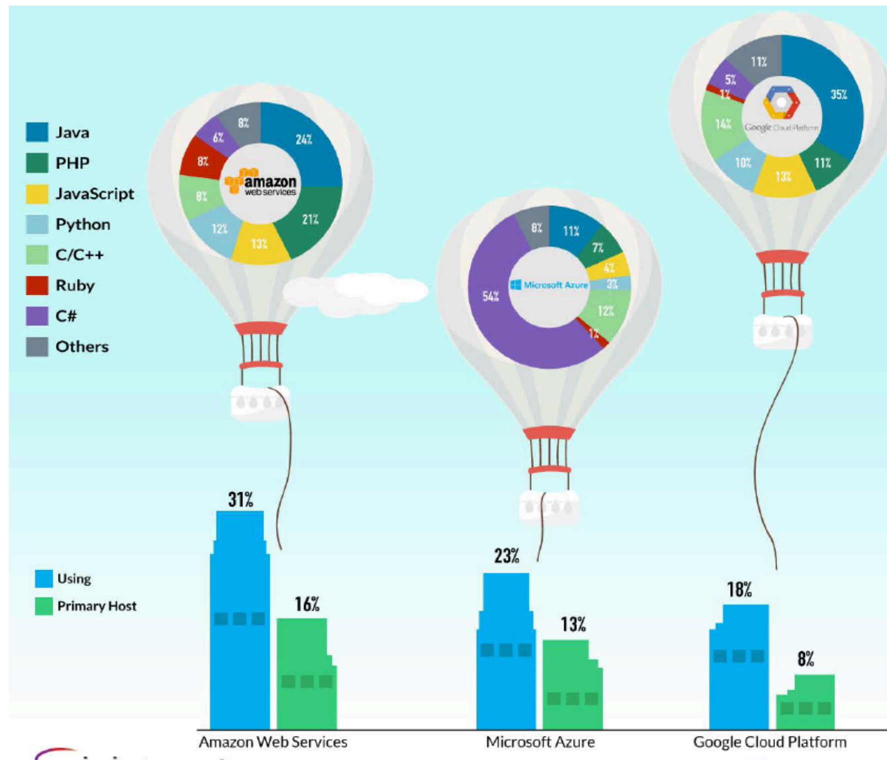




- Java 8 demonstrated that we have overcome and reversed overall perception of Java decline
- **Java 8 adoption is skyrocketing!**

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Java SE is #1 Runtime in the Cloud



- #1 Deployment runtime on AWS and Google App Engine and #3 on MS Azure
- Java Runtime is the foundation of the Cloud IaaS, PaaS and SaaS

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

ORACLE

Why Java Will Continue to Thrive in the Cloud

- Security & Vulnerability
  - Security built deep in the JVM runtime (bytecode verification, managed pointers & managed memory layout)
  - Security built in the language (strong typing and API encapsulation)
  - Upcoming hardware security innovations (from pages to pointer addresses, encryption at memory speed)
- Efficiency and ultimate performance
  - Dynamic JIT compilation (optimize and de-optimization JVM machinery, profile driven optimization)
  - Scalability (from 1 thread/core to 10,000s threads/cores, from a few GBs to multiple TBs)
  - Cloud is all about utility computing (operating cost will soon dominate Cloud market!)
- Increase deployment velocity
  - The cloud is demanding an accelerated deployment cycle (hour/day/week vs month/year)
  - Increase pressure to uptake new releases (security, critical fixes & performance) − manage backward compatibility!!!!
  - Thrive towards an homogeneous and uniform software infrastructure (lower maintenance cost)
- Manageability and serviceability at scale
  - Expressibility & Readability
  - Mature tooling ecosystems
  - Dynamic patching (no service downtime, class redefinition, JVM safepoint redefine)

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Challenges to Evolving The Java Platform

- 2 Trillion lines of code in production
  - Ensuring backward compatibility is a big deal to our users, to you and us :-)
- Decisions made 20 years ago may not apply anymore
  - Duality between Object and Primitive type
- Foreseeing the future
  - Many decisions made today may/will have profound impacts 10 years from now
- Maintaining the “look and feel” of Java

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

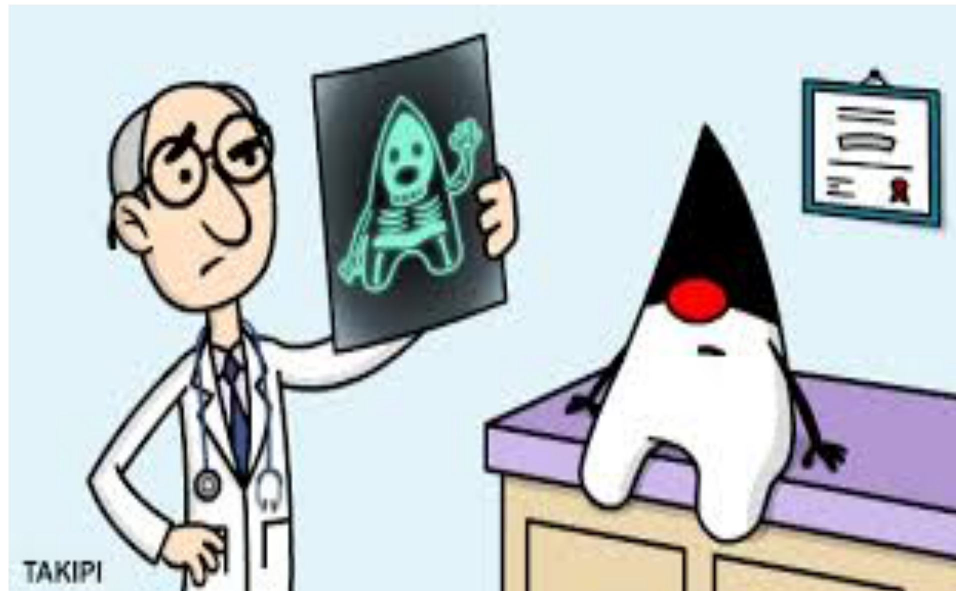
Java Platform Feature Backlog

"I have some paperwork to catch up. If I'm not back in two days, organize a search and rescue team!"

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Understanding Java Developers Main Pain Points



- Too verbose
- Use too much memory
- Too slow
- Dynamically linked
- Etc.

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Maintaining Java Core Principles

- Consistency
- Predictability
- Easy to read
- Compatibility

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Java SE Platform Investments

- Security is our #1 priority
  - Deliver the most secure Java runtime for the Cloud and next generation hardware
- Improving Java developer productivity and compatibility
  - Make Java developers more efficient and ensure easier path to uptake new releases quicker
  - Program once for the Cloud and scale up as needed
- Increasing density
  - Reducing memory usage and share more data between JVM processes
  - Largest impact to reducing our customers and our cloud operating cost
- Improving startup time
  - Enabling lighter-weight services and containers (microservice and Cloud devops use cases)
- Improving predictability
  - Lower, more predictable GC pauses, even for very large heaps (TB+)
- Simplifying serviceability and profiling
  - Managing and tuning JVM deployments at scale

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Java 9

"One Modular, Upgradeable Platform for the Cloud"

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Java 9 JEPs

- The big ticket item for java 9 is Modularization (aka Jigsaw)
- Currently 121 enhancements (JEPs)
  - 114 integrated, completed and closed
  - 7 in candidate, proposed to drop and targeted
- New EA build available every other week

http://openjdk.java.net/projects/jdk9/

102: Process API Updates
110: HTTP 2 Client
143: Improve Contended Locking
158: Unified JVM Logging
165: Compiler Control
193: Variable Handles
197: Segmented Code Cache
199: Smart Java Compilation, Phase Two
200: The Modular JDK
201: Modular Source Code
211: Elide Deprecation Warnings on Import Statements
212: Resolve Lint and Doclint Warnings
213: Milling Project Coin
214: Remove GC Combinations Deprecated in JDK 8
215: Tiered Attribution for javac
216: Process Import Statements Correctly
217: Annotations Pipeline 2.0
219: Datagram Transport Layer Security (DTLS)
220: Modular Run-Time Images
221: Simplified Doclet API
222: jshell: The Java Shell (Read-Eval-Print Loop)
223: New Version-String Scheme
224: HTML5 Javadoc
225: Javadoc Search
226: UTF-8 Property Files
227: Unicode 7.0
228: Add More Diagnostic Commands
229: Create PKCS12 Keystores by Default
231: Remove Launch-Time JRE Version Selection
232: Improve Secure Application Performance
233: Generate Run-Time Compiler Tests Automatically
235: Test Class-File Attributes Generated by javac
236: Parser API for Nashorn
237: Linux/AArch64 Port
238: Multi-Release JAR Files
240: Remove the JVM TI hprof Agent
241: Remove the jhat Tool
243: Java-Level JVM Compiler Interface
244: TLS Application-Layer Protocol Negotiation Extension
245: Validate JVM Command-Line Flag Arguments
246: Leverage CPU Instructions for GHASH and RSA
247: Compile for Older Platform Versions
248: Make G1 the Default Garbage Collector
249: OCSP Stapling for TLS
250: Store Interned Strings in CDS Archives
251: Multi-Resolution Images
252: Use CLDR Locale Data by Default
253: Prepare JavaFX UI Controls & CSS APIs for Modularization
254: Compact Strings
255: Merge Selected Xerces 2.11.0 Updates into JAXP
256: BeanInfo Annotations
257: Update JavaFX/Media to Newer Version of GStreamer
258: HarfBuzz Font-Layout Engine
259: Stack-Walking API
260: Encapsulate Most Internal APIs
261: Module System
262: TIFF Image I/O
263: HiDPI Graphics on Windows and Linux
264: Platform Logging API and Service
265: Marlin Graphics Renderer
266: More Concurrency Updates
267: Unicode 8.0
268: XML Catalogs
269: Convenience Factory Methods for Collections
270: Reserved Stack Areas for Critical Sections
271: Unified GC Logging
272: Platform-Specific Desktop Features
273: DRBG-Based SecureRandom Implementations
274: Enhanced Method Handles
275: Modular Java Application Packaging
276: Dynamic Linking of Language-Defined Object Models
277: Enhanced Deprecation
278: Additional Tests for Humongous Objects in G1
279: Improve Test-Failure Troubleshooting
280: Indify String Concatenation
281: HotSpot C++ Unit-Test Framework
282: jlink: The Java Linker
283: Enable GTK 3 on Linux
284: New HotSpot Build System
285: Spin-Wait Hints
287: SHA-3 Hash Algorithms
289: Deprecate the Applet API

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Java 9 Jigsaw Modularity

- Java SE & its implementations historically monolithic
  - Cloud deployments most often don't require the entire platform
- Backward compatibility requires strong API encapsulation
- Modules will define a `module-info.java` file
  - Declares its dependencies (requires)
  - Declares what packages it gives external access to (exports)
- Simple Example:

```
module com.foo.bar {
    requires com.foo.baz;
    exports com.foo.bar.alpha;
    exports com.foo.bar.beta;
}
```

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

JDK 9 Jigsaw Security:

Module boundaries enforced by the JVM

- Encapsulate implementation---internal classes inside modules
  - Share them with other implementation modules only as needed
- Massive maintainability improvement
- Simpler compatibility upgrade path
  - **We and You** can now hide and preclude access to unsupported internal APIs and implementation
- Will also significantly improve Security
  - Enable developers to create customized runtime that removed unused security sensitive APIs

 Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Java 9: Jigsaw New Linking Phase

- Introduce a new development phase: **Linking**

– After Javac compilation, but before packaging and deployment
  - – **`jlink`** linking tool to consume application classes as well as platform modules
  - – Linker can produce custom runtime images in various formats
    - Regular images (as today)
    - JVM---specific memory images (CDS archives)
    - "Fast executable binaries" (self---contained native executable code)

Demo

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.

Java 9: Ahead of Time (AOT) Java Compiler

- The unification of static and dynamic compilation
  - Static compilation – faster startup, lower memory usage, but limited in optimizing code generation
  - Dynamic profiling based compilation – slow startup but optimum code generation
- New AOT Compiler to statically compile Java classes to native shared libraries
  - Reduces startup time **and** improve density to close the gap against native service
- Compile Java packages to native shared libraries
- JVM was modified to load native shared libraries on startup
  - JVM internal structures, which describe compiled code, are split to describe compiled code in code cache and in a shared library
  - AOT compiled code is dynamically linked to Java methods after its class is initialized

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Java 9: Ahead of Time (AOT) Java Compiler

- New JDK tool 'jaotc' is added as part of java installation. 'jaotc' is java static compiler which produces native code for compiled java methods. It uses libelf for producing .so AOT libraries.

  To use 'jaotc' user have to specify .class, .jar files or java module name as input and resulting AOT library name as output:

  jaotc --output libHelloWorld.so HelloWorld.class

  jaotc --output libjava.base.so --module java.base

- User can specify which methods to compile or exclude with —compile-commands flag

  jaotc --output libjava.base.so —compile-commands base.txt —module java.base

- During JVM startup the AOT initialization code looks for shared libraries in ($JAVA_HOME/lib) or libraries specified by -XX:AOTLibrary option. If shared libraries are found, these are loaded and used.

  java -XX:AOTLibrary=./libHelloWorld.so,./libjava.base.so HelloWorld

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

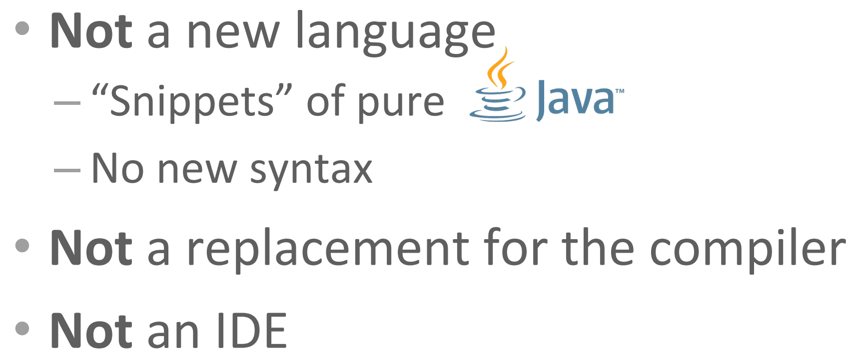
JDK 9 - What is JShell?

- Tool providing a dynamic interaction with the Java™ programming language in the Cloud
- Read-Evaluate-Print Loop (REPL) for the Java™ platform in the Cloud
  - Type in a snippet of Java code, see the results
- Deeply integrated with JDK tool-set
  - Stays current and compatible
- Also, an interactive API for use within other applications

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

What is JShell NOT?

- **Not** a new language
  - “Snippets” of pure 

  - No new syntax
- **Not** a replacement for the compiler
- **Not** an IDE

ORACLE®

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Who wants JShell?

- New to Java, new to programming
  - Start with expressions vs classes
  - Immediate feedback
- Exploring a new API or language feature
  - Experiment and instantly see results
- Prototyping and interactivity
  - Incrementally write complex code

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

With JShell

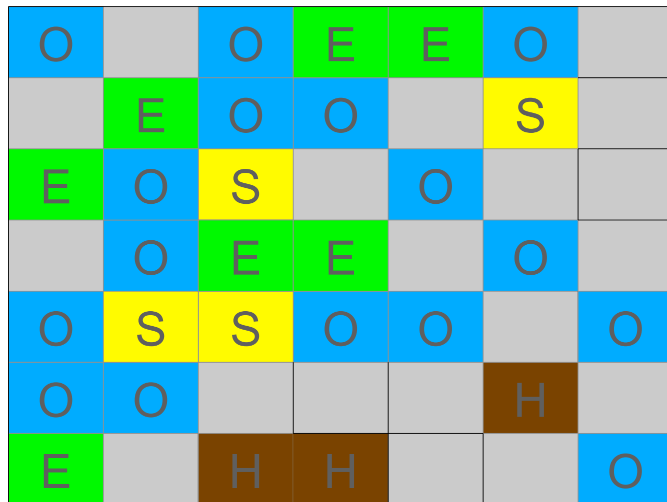
- Type in a “snippet” of code
- Immediately see its behavior

DEMO

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Java 9 - G1 Garbage Collector as the Default



E Eden regions
S Survivor regions
O Old generation regions
H Humongous regions
Available / Unused regions

- Multi-year effort to deliver next generation GC improvements
- GC pause-times have the largest impact on application performance, predictability and responsiveness
- G1 uses one contiguous heap space divided into many fixed size regions (analog to pages in VM OS's)
  - Per region scalable collection process
  - Size can be 1 MB – 32 MB
  - New architecture to scale up to multi-TB heap
- Each region can be assigned a unique eviction/ compaction policy (Eden region, Survivor region, Humongous or Old region)

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

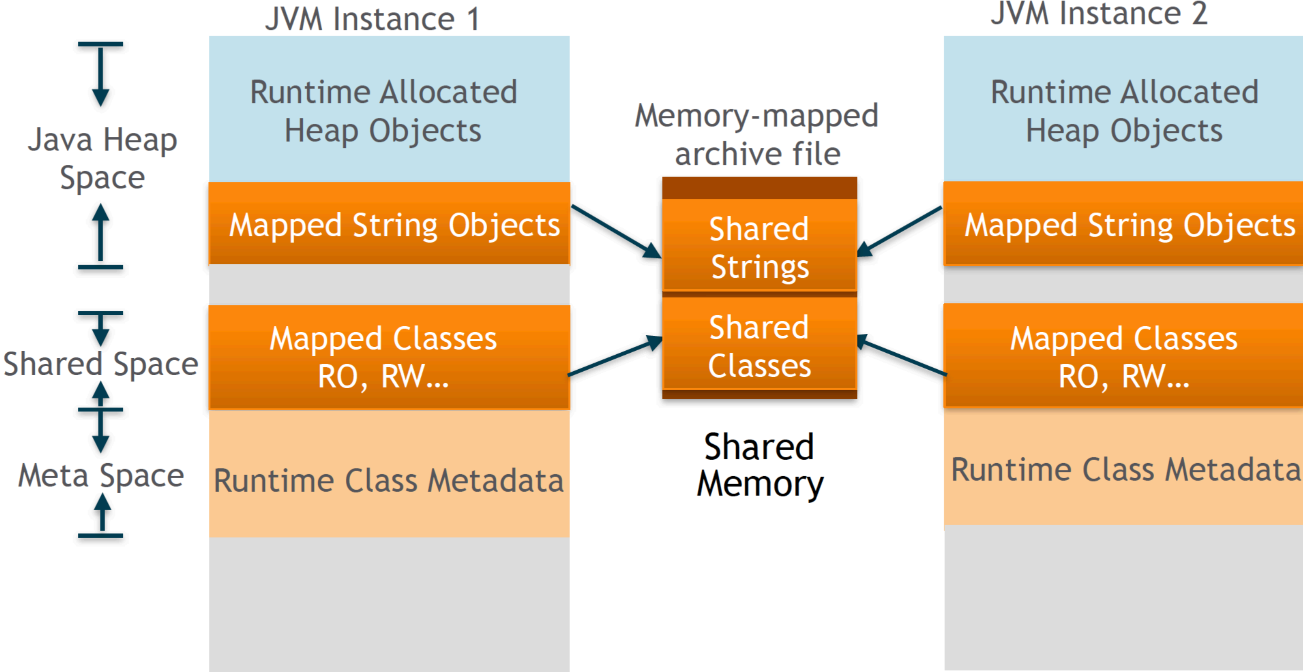
What is AppCDS?

- Commercial feature to reduces physical memory usage for loaded classes
- Faster startup time due to pre-loaded class metadata.
- Shares of class metadata across JVM processes.
- AppCDS stores class metadata in a Java Shared Archive file (JSA)
  - bytecodes, field tables, method tables, etc.
- JVM was modified to load more efficiently these Shared Archive files
- Multiple JVM processes can memory-map the same JSA file
  - Memory in the JSA file is shared across processes
  - Memory is split into Read-Only (shared) and Read-Write (Copy-on-Write)

ORACLE®

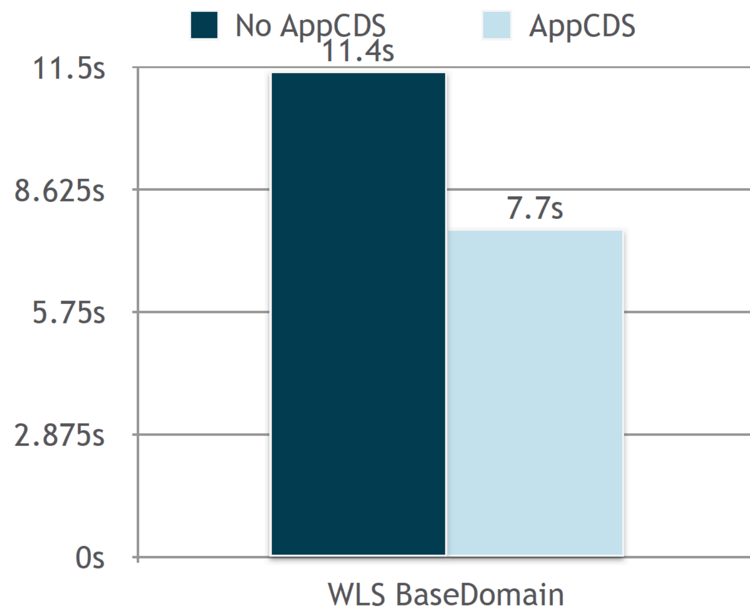
Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

AppCDS Architectural View



Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Startup Time Improvements for Sample Oracle WebLogic Server Environment



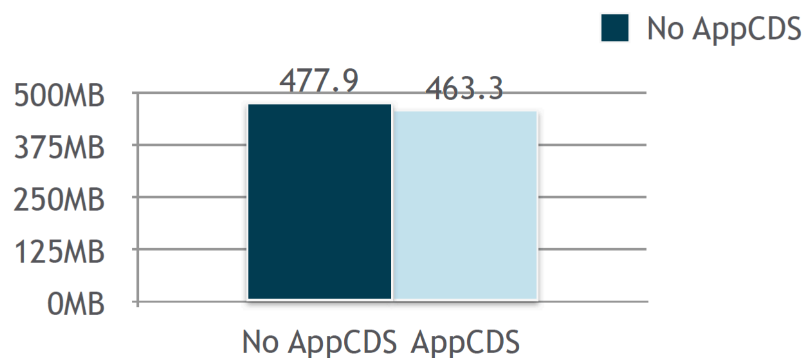
- About 30% startup time improvement observed with AppCDS for Oracle WebLogic Server (WLS) base domain

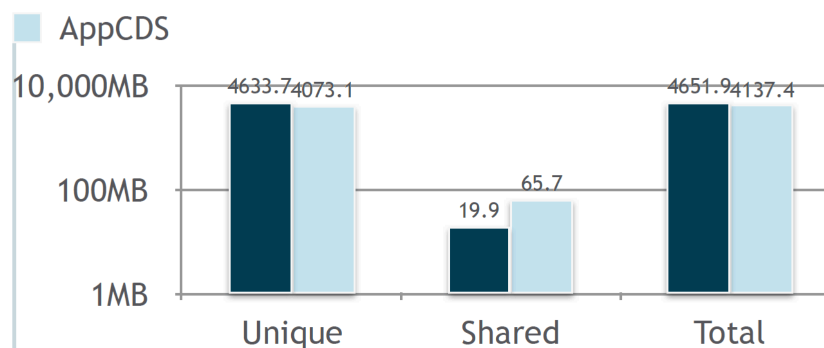
ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Memory Footprint Saving for Sample Oracle WebLogic Server Environment

No AppCDS   AppCDS





- With single JVM instance running WLS base domain, using AppCDS improves memory footprint
- Memory footprint improved by optimal layout and compaction of archived data

- With 10 JVM instances running WLS base domain simultaneously, shared memory increases
- There is 11.06% saving in total memory footprint with AppCDS

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

String Density

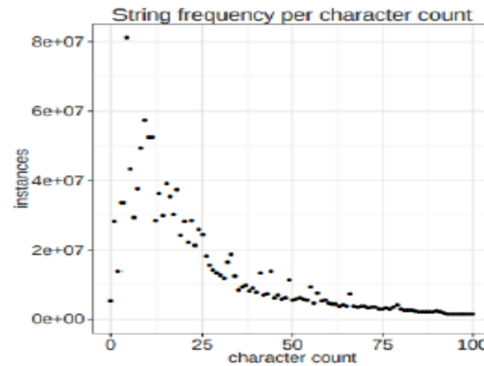
- The String Class stores characters using a UTF-16 encoding (use 2 bytes per character)
  - But, the vast majority of Java apps really use only single byte characters
- Consequence
  - Lots of wasted space / memory
- JDK 9 feature (JEP 254)
  - Modify the JVM runtime, JIT, GC and String class to transparently optimize the internal character encoding (use single vs multi-bytes)
  - Requirements
    - Reduce memory footprint, yet not sacrifice throughput performance
    - No API change, no re-compilation or code modifications required by upstream FMW or Fusion Apps to realize benefit
    - Support on X86/x64, SPARC and Linux, Solaris, Windows, OS X

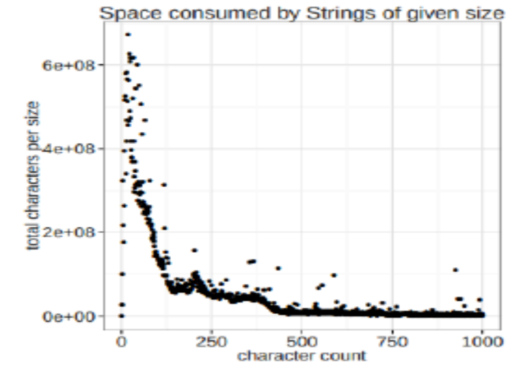
ORACLE

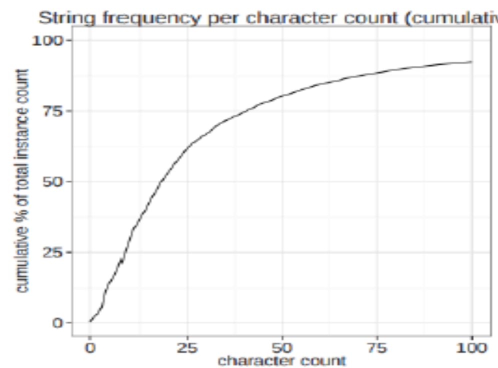
Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

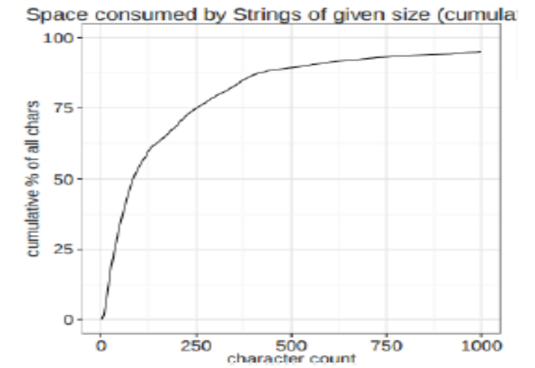
Density String

- Analyzed nearly 1,000's heap profiles
  - Vast majority of characters in Java Strings are single byte chars
  - 75% of Strings are smaller than 35 Characters
  - 75% of Characters are in Strings of Length < 250
  - 5% - 20% memory footprint reduction opportunity per application
- Specjbb 2005
  - **21% memory footprint reduction**
  - **27% less GCs**
  - **5% throughput improvement**









ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Flight Recorder Performance

Extremely Low Overhead

- Built into the JVM/JDK, by the people developing the JVM
- Less that 1% overhead in production deployment
- High performance flight recording engine and high performance data collection
  - Access to data already collected in the JVM runtime
  - Thread local native buffers
  - Invariant TSC for time stamping
  - More accurate method profiling (method profiling data even from outside safe-points)
  - Faster and more accurate allocation profiling

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Beyond 9

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

The Next Big Challenge: Object Data layout

- Java is very good at optimizing code, less so at optimizing data
  - Memory overhead, less than optimal performance, difficult to utilize modern hardware
- Java's type system gives us primitives, objects, and arrays
  - Very flexible! Can model almost anything.
- But flexibility is not exactly where we need it
  - Primitives are very rigid
  - Objects are more flexible than we always need
- The big problem: object identity
  - Needed for polymorphism, mutability
  - Not all objects need it, but all objects pay for it

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Improved Java/Native Interoperability

- Big Data Hadoop and Spark are highly dependent on native libraries
  - These dependencies won't go away
    - OS I/0 entries, GPU BLAS/LAPACK, AVX/Crypto/CRC intrinsics
  - Interacting with Native library from Java is hard at best
  - JNI is complex, slow and hard to secure - true for code, as well as for data
- Meanwhile, Java has significant technical debts in support of foreign calls
  - APIs export data structure layouts (struct stat) which are hard to traverse
  - The "cultural practices" (like safety) are different between Java and C
- Project Panama - provide an easier, safer and faster JNI
  - Adding Foreign Function and foreign data support to Java (Frozen Arrays, Vector API and Arrays 2.0)
  - Automatic type translation (native vs. carrier/java types)
  - No unsafe – provides safe APIs to access native data/functions

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

Summary

- Java SE adoption is thriving in the Cloud and will continue!
- Need your help on providing feedback on Java 9
  – https://jdk9.java.net/download/
- Beyond 9, we have a solid technical roadmap
- Let's continue to innovate and advance the Java SE Platform on OpenJDK together!

ORACLE

Copyright © 2016, Oracle and/or its affiliates. All rights reserved. |

ORACLE®