How to: make things start faster

Ludovic Henry
Java Engineering Group - Microsoft
About me

• Microsoft

• Empowering every Java developer to achieve more!

• Focus on VM Engineering
About you

• Who develops with the JDK daily?
  • Which languages?

• What kind of workload
  • Web services?
  • Big Data?
  • Desktop application?
  • Other?

• Which metrics are your primary focus
  • Throughput?
  • Latency?
  • Footprint?
  • Startup?  ➜ Focus for this talk
Agenda

Aside: The Importance of Methodology
Managing State
AOT Compilation and Beyond
If you have any questions, please interrupt me at any time!
Aside: the Importance of Methodology
The Project

- **Objective:** make the product more responsive

- **Key Result:** reduce time from *Cluster Creation* to *Session Ready*

- **Context:**
  - Distributed Services (3+ machines)
  - Mostly Java processes (15+)
  - Many processes per machine (5+)
Given that context, what would you do?
What we did: followed a Methodology
Methodology

• Java Performance Diagnostic Model (jPDM)
  • Using jPDM, a performance diagnostic model [Jfokus 2017, Kirk Pepperdine]

• Identify the bottlenecks
  • vmstat
  • GC logs
```sh
$> vmstat 1
procs -----------memory---------- --swap-- ----io---- --system-- --cpu-----
r  b  swpd free  buff  cache si  so  bi  bo  in  cs  us  sy  id  wa  st
11  3   0 16045660  9876 147268  0  0   3284 196 384 1088  4 19  73  4  0
 1  6   0 15932628 11024 235400  0  0   70736 640 1465 7377  7 12  11 70  0
 9  4   0 15850204 11736 305340  0  0   67780 508 1089 5284  5  6  26 63  0
17  1   0 15773448 13064 358904  0  0   46256 2692 2749 13125 33 16 26 24  0
 2  1   0 15697464 14432 405832  0  0   28952 3460 3476 16288 44 30 10 15  0
 1  2   0 15670680 15156 420480  0  0   9572 1780 2515 7377  7 12  36 24  0
14  1   0 15653340 15596 432096  0  0   7252 2048 2590 7254 23  9  43 25  0
 2  1   0 15628064 16096 443696  0  0   8312 2200 3452 11927 13 13  52 22  0
 1  1   0 15621172 16232 452180  0  0   8448 304  909 3315  3 12  65 19  0
 0  7   0 15239784 19472 509436  0  0  26160 1388 2547 8426 49  9  36  0  0
 1  5   0 15154380 20876 578928  0  0   22548  872 1903 5944 27  5  12 56  0
 0  5   0 15194716 20972 603288  0  0   18128  984 2037 6548 12  6  23 60  0
 4  3   0 15000088 21392 621432  0  0   10788  768 2407 7380 34  7  17 41  0
 0  4   0 14967924 21480 669056  0  0   7000  452 1130 3815 19  2  25 53  0
 1  4   0 14915884 21512 679332  0  0   9480 188 1109 3969 43  3  16 38  0
```

Launch

main()

Ready
Solution

• Prefetch files to filesystem cache
  • mmap + mlock from another process launched at boot

• Results
  • Avg: 51s to 43s ➔ 15% faster
  • 95p: 61s to 49s ➔ 20% faster
```
$> vmstat 1

procs -----------memory---------- --swap-- ----io--- --system-- --cpu--
  r  b  swpd free  buff  cache si  so  bi  bo  in  cs  us  sy  id  wa  st
 8 11 16042280 10444 153436  0  0  3456 218  394 1089  4 21 70  4  0
 0  9 15936404 11128 234228  0  0  69508 532 1149 5870  6 12  5 77  0
 0  4 15844404 11536 307684  0  0  69084 760 1917 6875  4  9 18  6  9
 0  5 15769280 12384 378260  0  0  69452 600 1270 4007  9  5 32  5  4
 0  5 15691104 12772 448248  0  0  69328 688 875 3335  7  4 37  5 20
 0  3 15612672 13700 518440  0  0  69604 708 718 2365  6  4 43  4  7
15  9 15483816 14112 606592  0  0  69496 1100 1832 8286  31 26  8  2  0
 2  5 15427128 14428 675460  0  0  68344 1288 2038 7367  3 12  9  7  0
 0  3 15324972 14884 751408  0  0  68984 1232 1350 4248  20  9 28  4  0
 0  3 15269128 15104 821236  0  0  69200 988 1303 3811  8  3 38  5  0
 1  3 15204932 15400 886728  0  0  64024 1372 1972 5628  19 10 26  4  0
 1  2 15156048 16244 903088  0  0  7572 4696 5054 20494  32 33 25  1  0
 2  0 15171516 16484 905324  0  0  1652  848 2206 6039  15  8 69  7  0
 1  0 15171980 16516 904796  0  0  0  84  238 1089  0  1 99  0  0
 3  0 15146488 16584 911560  0  0  3160  316 815 3028  12  4 81  3  0
10  4 14990868 19740 945696  0  0  11804 5324 2614 18250  66 25  7  2  0
 8  3 14853528 20612 976652  0  0  23616 1660 1284 10272  84 13  0  3  0
10  4 14748248 20972 997824  0  0  16432 2040 2373 23566  79 18  1  3  0
13  3 14707516 21076 1021824  0  0  21884 848 2643 32709  85 12  1  2  0
 5  4 14675788 21224 1038060  0  0  13904 1480 11513 51637  73 15  4  9  0
 5  3 14625984 21332 1055720  0  0  13004 888 22791 70219  52 18  9 21  0
 2  3 14601044 21436 1065760  0  0  7152 732 7463 23289  40  8 18 34  0
```

Launch

main()

Ready
```
Managing State
What State?

• In the OS
  • Filesystem Cache
  • File Descriptors (e.g. files, sockets, devices)

• In the JVM
  • Metadata
  • Code

• In the Application
  • Data
The image illustrates the lifetime of an application in a Java development environment. The lifecycle is divided into three main phases: Start-up, Ramp-up, and Steady-State.

**Start-up**:
- JVM Initialization
- Class Loading
- Code Generation
- Class Initialization
- Class Loading

**Ramp-up**:
- Class Loading
- Code Generation
- Class Initialization
- ...

**Steady-State**:
- Application at Steady-State

The diagram shows the flow of the application lifecycle, starting with the JVM initialization and proceeding through the loading of classes and code generation until reaching the steady-state where the application's main() method is executed.

- **CDS** (Compiled Directly to System) and **AOT** (Ahead Of Time) are used at different stages of the application lifecycle.
- Checkpointing is used to save the application's state (data, sockets, files, etc.) for later recovery.

The DevDiv Java Engineering Group logo is visible in the bottom right corner of the image.
native-image / SubstrateVM

- AOT Initialization
  - Heap Snapshotting
  - Class Initialization

- AOT Compilation
AOT Compilation and Beyond

In HotSpot, not native-image / SubstrateVM
AOT Compiler in HotSpot

Available in Java 9+

Based on Graal JIT compiler

https://docs.oracle.com/en/java/javase/13/docs/specs/man/jaotc.html

- --class-name / --jar / --module / --directory
- --compile-command
- --compile-for-tiered
- -Jflag
Tradeoffs

Position Independent Code (PIC)

• Foundational concept of Shared Libraries

https://www.technovelty.org/linux/plt-and-got-the-key-to-code-sharing-and-dynamic-libraries.html

What’s going on here? What is happening is lazy binding — by convention when the dynamic linker loads a library, it will put an identifier and resolution function into known places in the GOT. Therefore, what happens is roughly this: on the first call of a function, it falls through to call the default stub, which loads the identifier and calls into the dynamic linker, which at that point has enough information to figure out "hey, this libtest.so is trying to find the function foo". It will go ahead and find it, and then patch the address into the GOT such that the next time the original PLT entry is called, it will load the actual address of the function, rather than the lookup stub. Ingenious!

• AOT requires indirect access to symbols, JIT doesn’t
Tradeoffs

Missed Optimizations: dead-code elimination

```java
final static boolean shouldExplode = rand.nextBoolean(); // hopefully not!

static void tryExplode()
{
    if (shouldExplode)
    {
        explode();
    }
}
```
Tradeoffs

Missed Optimizations: lack of Profiling Data

• It doesn’t know what types are used at call sites
  • Inlining

• It doesn’t know which code is cold/warm/hot
  • Basic Block Ordering
  • Register Allocation
Integration to Tiered Compilation Pipeline
Integration to Tiered Compilation Pipeline
Demo

https://github.com/luhenry/demo/tree/master/java-aot
Class Data Sharing

- JDK 5: Class-Data Sharing
- JDK 10: Application Class-Data Sharing
- JDK 12: Default Class-Data Sharing Archives
- JDK 13: Dynamic Class-Data Sharing
Demo

https://github.com/luhenry/demo/tree/master/java-cds
What’s missing in HotSpot?

• Better integration between components: jlink, jaotc, CDS
  • Lots to learn from native-image / SubstrateVM
    • AOT Initialization beyond string in CDS

• Better integration to Tiered Compilation pipeline
  • Integrate at Tier-3 (C1 w/ full profile)
    • No need for Interpreter
    • Deoptimization from C2 to AOT
    • Allows for JIT-less runtimes with JITaaS
Thank you!

luhenry@microsoft.com
java-performance@microsoft.com

Ludovic Henry
Java Engineering Group - Microsoft