OpenJDK Architecture

Andrew Dinn
Red Hat Open Source Java Team

March 2014

Architecure & Design through the Code Base

- Up Front Health Warning
- This talk is very code-oriented
  - Will include mention of
    - locations in the code base
    - functions/methods and classes/types
- Point is to kill 3 birds with one stone
  - Why is OpenJDK built the way it is?
  - How is OpenJDK built the way it is?
  - Where is OpenJDK built the way it is?
  - i.e. Familiarization for the purpose of *hacking*

OpenJDK = JDK + JVM

- JDK Class Library
  - Java code & native C/C++ libraries
    - jre classes
      - deployed in jre/lib/rt.jar + ...
    - sdk classes
      - deployed in lib/tools.jar + ...
- Hotspot JVM
  - Compiled C++ code
    - Bootstrap into Java execution
    - Virtualize underlying OS/cpu
      - threads, memory, io, JIT, etc
    - deployed in jre/lib/<arch>/libjvm.so

JDK CLass Library

OpenJDK = lots of JDK code + JVM

- JDK Class Library
  - big and still growing
- 5 sub-repos of Java code & native libraries
  - jdk
    - mostly jre classes (bootstrap, system, libraries)
    - a few sdk classes (e.g. jvmti support)
    - OS-specific subclasses – e.g. awt, Process, FileSystem etc
  - langtools
    - only sdk classes (javac, javadoc, etc)
  - corba, jaxp, jaxws
    - wt??? really Java EE not SE

JDK <--> Hotspot Interface

jdk <-> hotspot interface

- API mostly functions declared as JVM_ENTRY
  - conventionally named JVM_Xxx
    - e.g. JVM_StartThread
    - can be called from JDK native method implementation
    - but . . .
- jnienv method RegisterNatives
  - native method fastpath to JVM_ENTRY function
  - called by class static init . . .
    - Thread, Compiler, Object, Class, System, ClassLoader etc

jdk launcher <-> hotspot interface

- launcher provided by jdk
  - in src/share/bin/java.[h/c]
  - used by java, javac etc
- small bootstrap API provided by libjvm
  - in src/share/vm/prims/jni.[h/cpp]
    - JNI_CreateJavaVM
    - . . .
  - launcher also accesses VM functions via callbacks in
    - struct JavaVM
    - struct JNIEnv

Hotspot

OpenJDK = mostly hotspot (most interestingly)

- just single hotspot sub-repo
  - almost entirely C++ code
- ~90% generic (arch-neutral)
  - src/share/vm/<function>
    - each functional subdir is a src tree and include root
  - src/share/tools/<tool>
    - not part of JVM per se
      - libhsdis.so uses binutils to disassemble code

hotspot = many OS and cpu combinations

- code factored by os and/or cpu
  - src/os/<os>/vm,
  - src/cpu/<cpu>/vm,
  - src/os_cpu/<os>_<cpu>/vm
    - all are both src trees and include roots
- os includes
  - Windows, Linux, Solaris, AIX, BSD, OSX
- cpu includes
  - x86(_32/64), AArch64, PPC, Sparc, zero**
- os_cpu inlcudes a *sparse* cross--product

Hotspot: shared code

hotspot: utility code

- many utility classes
  - general purpose in separate dirs – libadt, utilities
  - more specialized with client code – runtime/timer
- n.b. src/share/vm/utilities/debug.[hpp/cpp]
  - call these functions from gdb
    - find method for pc
    - print stack
    - dump threads, etc

hotspot: oops – Java data & metadata

- see src/share/vm/oops/oops.* oopsHierarchy.*
  - oopDesc == C++ overlay for any Java object

```
class oopDesc {
    markOop _mark;
    Klass* _klass;
}
```

  - oop == [C++ accessor for] Java reference
    - typedef class oopDesc* oop
  - debug builds override operations via methods

```
class oop {
    oopDesc *o;
    bool operator == (void *) . . .
    operator oopDesc*() . . .
}
```

hotspot: oops hierarchy

- oop & oopDesc have a hierarchy of subclasses

  oop

      instanceOop

      arrayOop

          objArrayOop

          typeArrayOop

  typedef xxxOopDesc* xxxOop

- also a couple of related types

  markOop

  - header element overlay for GC and lock operations

  narrowOop

  - special for when -XX:+UseCompressedOops
  - expands 32 bit oop into 64 bit object address

hotspot: oops – metadata Klass hierarchy

- Klass -- models Java class as C++ type
  - Klass
    - InstanceKlass
      - InstanceClassLoaderKlass
      - InstanceMirrorKlass
        - (for java.lang.Class instances)
      - InstanceRefKlass
    - ArrayKlass
      - ObjArrayKlass
      - TypeArrayKlass
  - narrowKlass

hotspot: runtime support layer

- in src/share/vm/runtime
- global configuration
  - i.e. -XX:[+/-]GlobalConfigVar[=value]
    - in globals.hpp
- execution support functions/types
  - locks, Java/VM threads, stack frames
  - handles (== GC-visible oop slot)
  - generic management of JITted stubs
    - see esp. sharedruntime.[hpp/cpp]
    - Java -> C++, Java --> Java link routines
    - C++ ineffables (e.g. cache flush)

hotspot: memory management

- utility classes and API definitions
  - in src/share/vm/memory & gc_interface
  - regions, chunks, free lists, barriers, card tables
  - reference processing
- specific implementations
  - under src/share/vm/gc_implementation
  - shared subdir
    - spaces & buffers, timers & counters, GC threads/policies
  - CMS, G1, Parallel, ParNew

hotspot : GC implementations CMS

- Concurrent Mark Sweep
  - Genarational GC
    - ParNew Young Gen
      - Eden + Pair of Survivor Spaces
    - Mark Sweep Old Gen
      - mostly concurrent
      - sweep to free lists
  - Fragmentation a problem
    - falls back to stop-the world serial compaction
  - Card Table a Problem
    - tracks Old -> Young Gen references
    - card mark can introduce cache contention

hotspot : GC implementations G1

- Garbage First
  - Generational
    - ParNew Young Gen
  - Region Based Old Gen Management
    - evacuate from most empty regions
    - compacts as it relocates
  - Large objects an issue
    - need to evacuate contiguous regions
  - Remembered sets a problem
    - remembered sets track inter-region refs
    - can be very large and can introduce cache contention

hotspot: interpreter(s)

- in src/share/vm/interpreter
- C++ Interpreter
  - conventional inner loop case switch interpreter
  - slow but easy to port
- Template Interpreter
  - dispatch table of 'per-bytecode' generated asm
    - Java stack <== machine stack
    - generated asm manipulates stack and/or VM state
    - dedicated machine registers for method & bytecode pointer
    - asm epilog increments bytecode and dispatches
  - '10x' faster than C++ interpreter

hotspot: runtime machine code generation

- in src/share/vm/asm & code
  - generic register & assembler classes
    - Register
      - cpu-dependent code defines actual register set
    - AbstractAssembler
      - cpu-dependent subclasses, Assembler, MacroAssembler etc
  - instruction patching
    - needed for dynamic call resolution & deopt
  - code management
    - buffers, blobs,
    - relocs, debug info
    - stub methods, compiled methods

hotspot: compiler interface

- in src/share/vm/compiler
  - compilation driver
    - API to queue requests
    - dedicated compiler threads
- in src/share/vm/ci
  - compiler <--> vm abstraction layer
    - limits compiler's knowledge of vm

Andrew Dinn

hotspot: compilers C1

- client compiler
  - traditional optimizing compiler
    - good code
    - fast compilation
- good for desktop client apps
  - hardcore optimizing JIT would be JTL (Just Too Late)
- also used for -XX:+TieredCompilation
  - interpret (gather profile info) ==>
  - c1 compile (gather profile info) ==>
  - c2 compile

hotspot: compilers C2

- in src/share/vm/opto
- server compiler
  - *highly* performant code
  - slower but still o(n log(n)) time for n bytecodes
- parses bytecode to ideal graph
  - most optimization at ideal level
    - main optimization scheme based on GCM/GVN (Click 95)
      - GVN provides highly efficient SSA data representation
      - combines control, dataflow, io and memory dependencies
      - type lattice supports very aggressive optimizations
    - some ad hoc graph rewriting

hotspot: compilers C2 back end

- in src/share/vm/adlc
- architecture description language compiler
  - lowering, scheduling,
    code generation, peephole optimization
- each per cpu back end provides ad file
  - register model
    - drives generic register allocator
  - lowering rules
    - matcher translates ideal node/subgraph --> insn (sequence)
  - insns linked to cost & pipeline model
    - scheduler tries to minimise cost & delays

C2 Compiler Algorithms

- Global Code Motion / Global Value Numbering, Cliff Click. ACM PLDI 95
- A Fast Algorithm for Finding Dominators in a Flowgraph, Thomas Lengauer and Robert Tarjan, TOPLAS 79
- Register Allocation & Spilling via Graph Coloring, G J Chaitin, SIGPLAN 82
- Escape Analysis for Java, Jong Deok-Choi, Manish Gupta et al, OOPSLA 99

Hotspot: os- & os_cpu-dependent

hotspot os-dependent: examples

- os-specific global configuration
  - e.g. -XX:+UseTransparentHugePages
- signal handling
- mutexes & threads
- scheduling
- page & stack management
- timers & clocks

hotspot os_cpu-dependent: examples

- thread_local storage
- atomic load/store/xchg
- byte swap & copy
- thread stack management
- some signal handling (register 'fixing')

Hotspot: cpu-dependent

hotspot cpu-dependent: register model

- n.b. *all* cpu-dependent code in src/cpu/<arch>/vm
- register model
  - register_definitions_<arch>.*, register_<arch>.*
    - generic register declarations/definitions
  - vmreg_<arch>.*
    - cpu-specific register implementation

hotspot cpu-dependent: code assembly

- assembler_<arch>.*
  - encode cpu instruction set
- macroassembler_<arch>.*
  - encode logical ops as insn sequence
- interp_masm_<arch>.*
  - extend masm with extra ops for interpreter only
- nativeInst_<arch>.*
  - implement insn patching

hotspot cpu-dependent: runtime

- sharedRuntime_<arch>.*
  - generate Java --> C++ transition stubs
    - argument marshalling
    - register save/restore
    - native wrapper code
  - generate Java -> Java transition stubs
    - i2c/c2i stubs
    - exception_blob & handler_blob
    - deopt_blob & uncommon_path_blob
    - resolve_blob

hotspot cpu-dependent: runtime

- stubGenerator_<arch>.*
  - generates . . .
    - call stub (C++ --> Java)
    - catch unhandled excpn (C++ <-- Java)
    - forward_exception (Java <-- C++)
    - housekeeping stubs
      - atomic_xchg, atomic_cmpxchg, atomic_add
      - fences & memory barriers
      - stack walking
      - special case math code
      - inline copy

hotspot cpu-dependent: template interpreter

- templateTable_<arch>.*
  - methods to generate templates
    - one method per bytecode insn

```
void TemplateTable::dup() {
    // stack ... a
  _masm.load_ptr(0, rax);     // plant stack load
  _masp.push_ptr(rax);        // plant stack push
}   // stack: ..., a, a
```

  - methods to generate inline auxiliary code
    - e.g. resolve class or member, initialize classpool constant
      - prepare_invoke()
      - load_field_cp_cache_entry

hotspot cpu-dependent: template interpreter

- templateInterpreterGenerator_<arch>.*
  - methods to generate interpreter-specific stubs
    - normal call frame setup
    - native call frame setup
    - exception handling
    - exception throwing
      - including special exception throw cases
        - array bounds
        - class cast . . .
  - used where templates require special case handling
    - plant load and jump to stub

hotspot cpu-dependent: c1 implementation

- whole host of c1_Xxx files including
  - global config
    - c1_globals_<arch>.hpp
  - its own LIR and LIR optimizer
    - c1_LIRGenerator_<arch>.cpp
    - c1_LIRAssembler_<arch>.cpp
  - register allocator
    - c1_LinearScan_<arch>.cpp
  - assembler and runtime support
    - c1_MacroAssembler_<arch>.cpp
    - c1_Runtime_<arch>.cpp
    - c1_CodeStubs_<arch>.cpp

hotspot cpu-dependent: c2 implementation

- very few files – code mostly generated by adlc
  - global config
    - c2_globals_<arch>.hpp
  - declarative architecture description (very large)
    - <arch>.ad
      - registers & register classes
      - encodings
      - frame layout & calling convention
      - processor pipeline model
      - operand and instruction matching rules
      - peephole optimization matching rules
      - inline code
    - useful docn in ad files – helps to compare across ports

Questions?