ORACLE

Understanding the Native Image Build Process

FOSDEM 2025

Tom Shull

February 1, 2025

About Me

- Principal Researcher on GraalVM Project
- Joined Oracle Labs in 2020
- Work primarily on Native Image Team

Copyright © 2025, Oracle and/or its affiliates

Outline

(Brief) Native-Image Overview

Describe high-level stages of builder

More details about front end of builder

- Transformation of Java bytecode to Graal Graphs
- Interacting with Substrate World and Host JVM
  - Substrate World: Metadata, Methods, Objects installed within final executable
  - Host JVM: JVM running Native Image Builder

Copyright © 2025, Oracle and/or its affiliates

Goals

Better Understanding of

1. What phases exist within the builder
2. How to retrieve and view Graal Graphs
3. How the “Substrate Universe” is represented
4. How the “Substrate Universe” relates to the “Java Universe” of the Host JVM

Copyright © 2025, Oracle and/or its affiliates

Native Image Overview

Tool for creating standalone executables from Java Applications

- Ahead of Time (AOT) compile all “reachable” Java code
- Eagerly initialize and store some Java objects into “Image Heap”

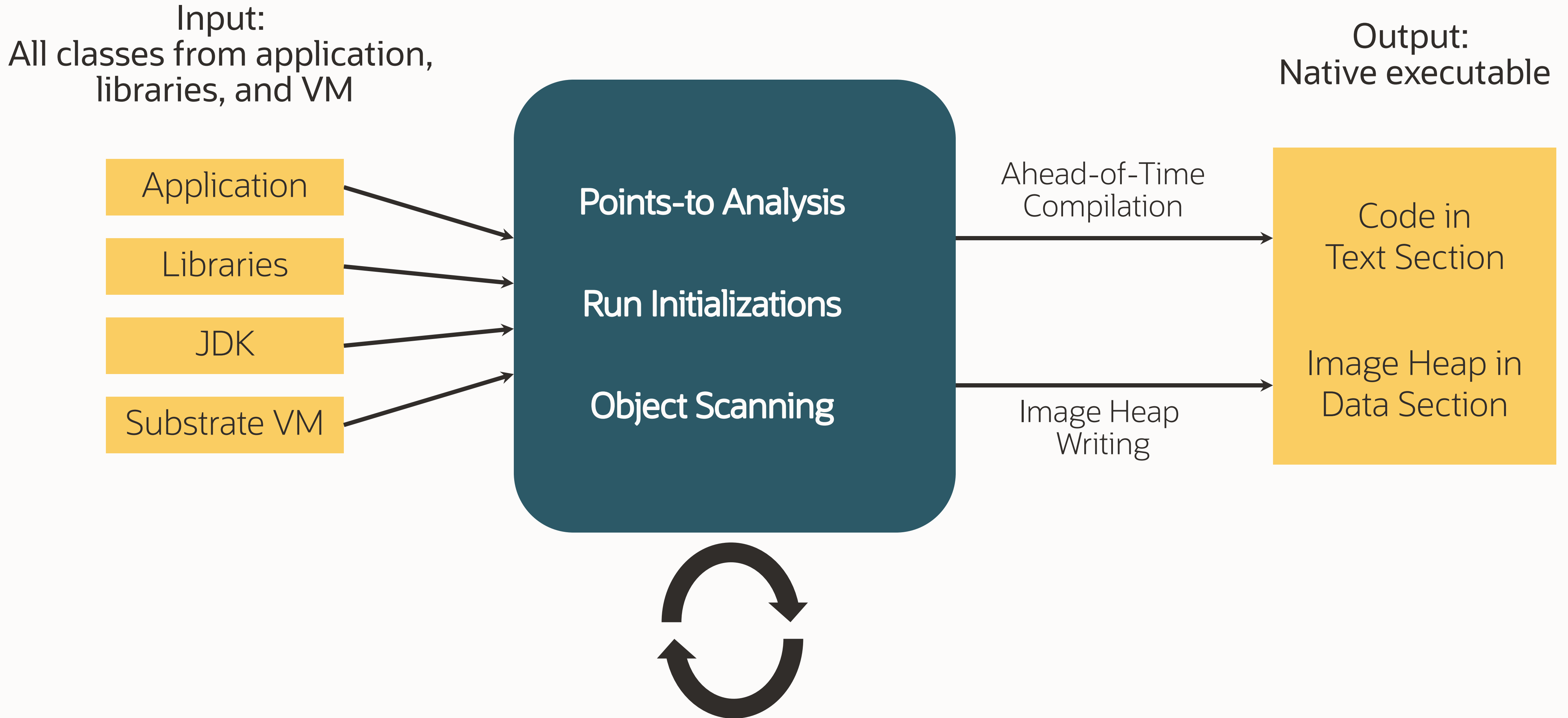
Benefits

- No JDK distribution needed at runtime
- Fast startup times
- Low memory footprint
- Predicable performance

Supports Spring Boot, Micronaut, Quarkus

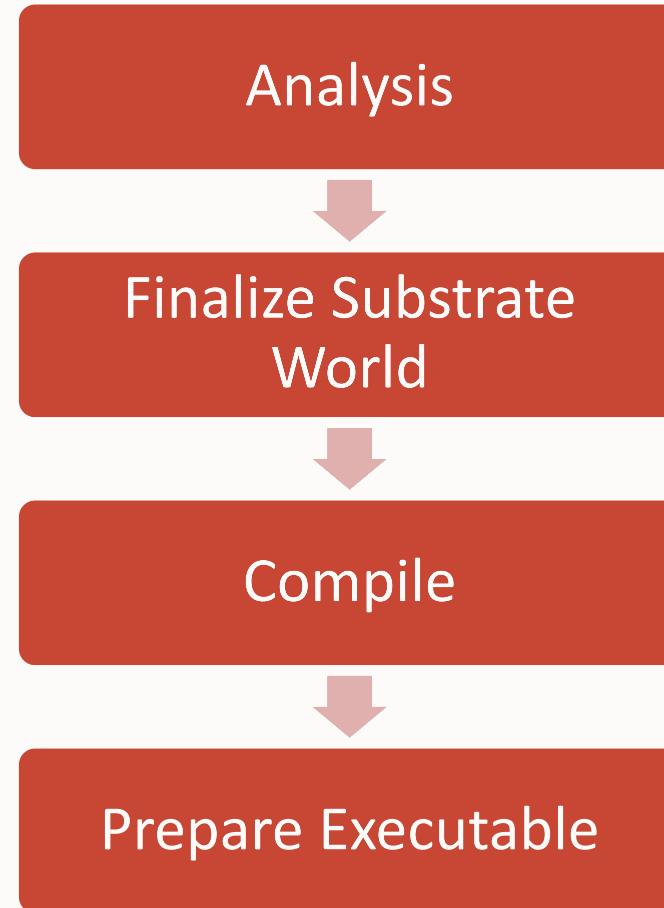
Copyright © 2025, Oracle and/or its affiliates

Native Image Builder Process



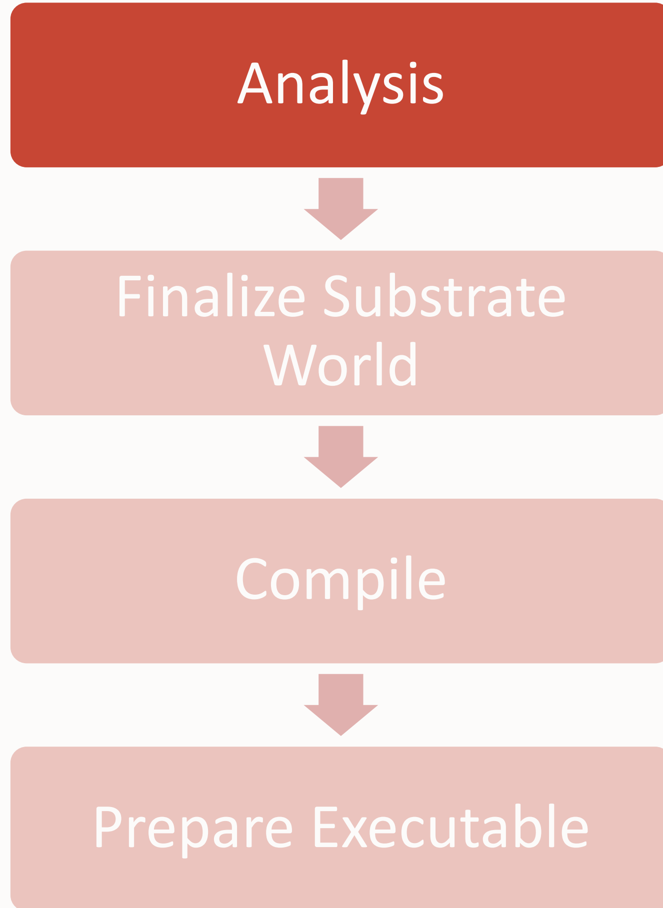
Copyright © 2025, Oracle and/or its affiliates

Builder Stages



Copyright © 2025, Oracle and/or its affiliates

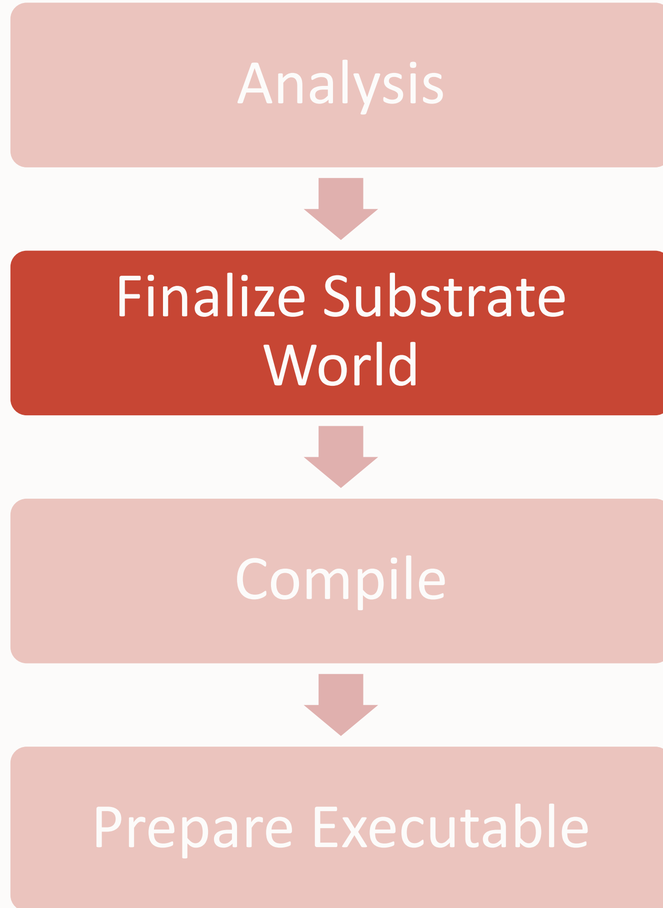
Builder Stages – Analysis



- Parse Java bytecode into Graal graphs
- Run Class initialization code
- Discover all potential Substrate World information

Copyright © 2025, Oracle and/or its affiliates

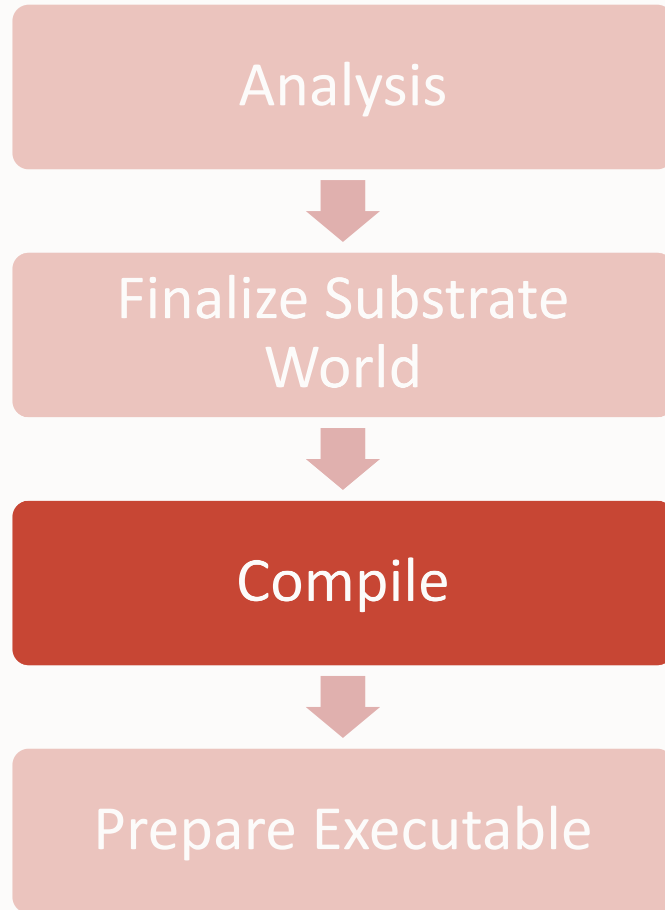
Builder Stages – Finalize Substrate World



- Finalize object layout, header information
- Optimize graphs based on analysis

Copyright © 2025, Oracle and/or its affiliates

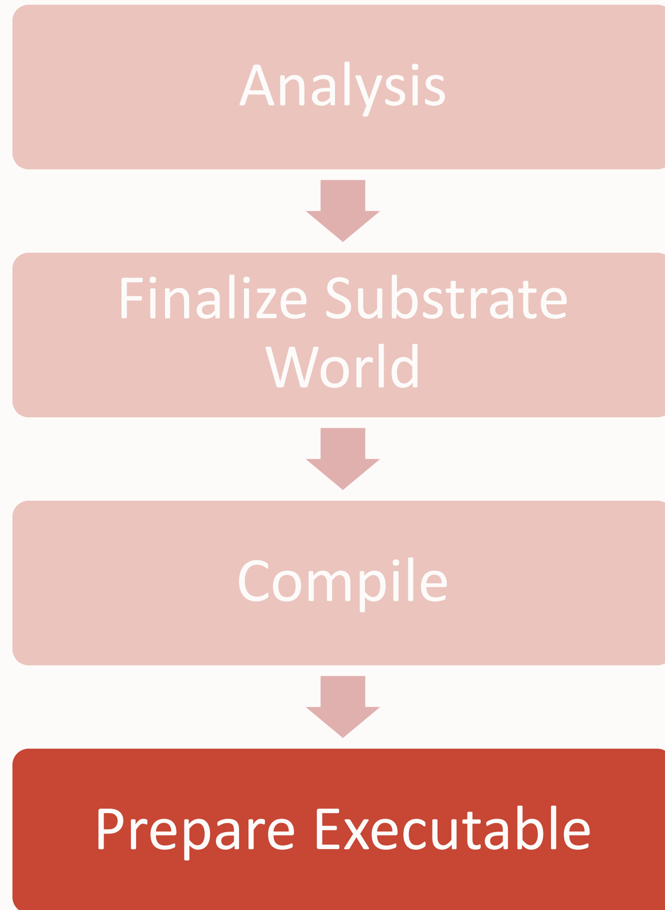
Builder Stages – Compile



- Compile graphs down to machine code

 Copyright © 2025, Oracle and/or its affiliates

Builder Stages – Prepare Executable



- Write Image Heap objects to data section
- Layout methods
- Resolve/Install relocations & invoke linker

Copyright © 2025, Oracle and/or its affiliates

Mapping Stages to Builder Output

```
========================================================================================
GraalVM Native Image: Generating 'hellofosdem' (executable)...
========================================================================================
[1/8] Initializing...                                                  (5.1s @ 0.12GB)
 Java version: 21.0.2+13, vendor version: GraalVM CE 21.0.2-dev+13.1
 Graal compiler: optimization level: b, target machine: armv8.1-a
 C compiler: cc (apple, arm64, 16.0.0)
 Garbage collector: Serial GC (max heap size: 80% of RAM)
 1 user-specific feature(s):
 - com.oracle.svm.thirdparty.gson.GsonFeature
----------------------------------------------------------------------------------------
Build resources:
 - 12.98GB of memory (75.6% of system memory, less than 8GB of memory available)
 - 8 thread(s) (100.0% of 8 available processor(s), determined at start)
[2/8] Performing analysis...  [*****]                                  (3.8s @ 0.46GB)
    3,113 reachable types   (67.5% of    4,613 total)
    3,585 reachable fields  (42.8% of    8,368 total)
   14,192 reachable methods (41.6% of   34,146 total)
      989 types,    13 fields, and   142 methods registered for reflection
       57 types,    56 fields, and    52 methods registered for JNI access
        4 native libraries: -framework Foundation, dl, pthread, z
[3/8] Building universe...                                             (1.1s @ 0.62GB)
[4/8] Parsing methods...      [*]                                      (0.6s @ 0.47GB)
[5/8] Inlining methods...     [***]                                    (0.5s @ 0.55GB)
[6/8] Compiling methods...    [**]                                     (2.3s @ 0.88GB)
[7/8] Laying out methods...   [*]                                      (1.3s @ 0.62GB)
[8/8] Creating image...       [*]                                      (1.6s @ 0.60GB)
   4.72MB (39.12%) for code area:     8,083 compilation units
   6.95MB (57.62%) for image heap:   85,017 objects and 55 resources
 392.68kB ( 3.26%) for other data
  12.06MB in total
----------------------------------------------------------------------------------------
```

Copyright © 2025, Oracle and/or its affiliates

Mapping Stages to Builder Output

```
================================================================================
GraalVM Native Image: Generating 'hellofosdem' (executable)...
================================================================================
[1/8] Initializing...                                          (5.1s @ 0.12GB)
 Java version: 21.0.2+13, vendor version: GraalVM CE 21.0.2-dev+13.1
 Graal compiler: optimization level: b, target machine: armv8.1-a
 C compiler: cc (apple, arm64, 16.0.0)
 Garbage collector: Serial GC (max heap size: 80% of RAM)
 1 user-specific feature(s):
 - com.oracle.svm.thirdparty.gson.GsonFeature
--------------------------------------------------------------------------------
Build resources:
 - 12.98GB of memory (75.6% of system memory, less than 8GB of memory available)
 - 8 thread(s) (100.0% of 8 available processor(s), determined at start)
[2/8] Performing analysis...  [*****]                          (3.8s @ 0.46GB)
    3,113 reachable types   (67.5% of    4,613 total)
    3,585 reachable fields  (42.8% of    8,368 total)
   14,192 reachable methods (41.6% of   34,146 total)
      989 types,    13 fields, and   142 methods registered for reflection
       57 types,    56 fields, and    52 methods registered for JNI access
        4 native libraries: -framework Foundation, dl, pthread, z
[3/8] Building universe...                                     (1.1s @ 0.62GB)
[4/8] Parsing methods...      [*]                              (0.6s @ 0.47GB)
[5/8] Inlining methods...     [***]                            (0.5s @ 0.55GB)
[6/8] Compiling methods...    [**]                             (2.3s @ 0.88GB)
[7/8] Laying out methods...   [*]                              (1.3s @ 0.62GB)
[8/8] Creating image...       [*]                              (1.6s @ 0.60GB)
   4.72MB (39.12%) for code area:     8,083 compilation units
   6.95MB (57.62%) for image heap:   85,017 objects and 55 resources
 392.68kB ( 3.26%) for other data
  12.06MB in total
--------------------------------------------------------------------------------
```

Analysis

Finalize Substrate World

Compile

Prepare Executable

Copyright © 2025, Oracle and/or its affiliates

Graph Creation

BytecodeParser: creates Graal Graphs from bytecode

- Performed once per method

Graph is ground source of truth

- Bytecode is only used for creating initial graphs
- Analysis directly reads graph

We can optimize graphs throughout entire build process

- Including before analysis

Copyright © 2025, Oracle and/or its affiliates

Graph Example

Start
Param b
Param a
<
If
Begin
Begin
End
End
Merge
Phi
Param sum
+
Return

```
int addMin(int a, int b, int sum) {
  int min;
  if (a > b) {
    min = b;
  } else {
    min = a;
  }
  return min + sum;
}
```


Copyright © 2025, Oracle and/or its affiliates

IGV

IGV (Ideal Graph Visualizer): tool for viewing Graal Graphs

Can view graphs before/after each phase

Important Native Image Flags

- -H:PrintGraph=[Network|File]
- -H:MethodFilter=[methods]
- -H:Dump=[dump level / phases]

See also `native-image --expert-options-all`

Copyright © 2025, Oracle and/or its affiliates

IGV Graph Phase Example

Outline | Source Collections

Dump 29628 (Sun Jan 26 12:09:35 CET 2025)
- HelloFOSDEM.main(String[])
- HelloFOSDEM.main(String[])
- HelloFOSDEM.main(String[])
- HelloFOSDEM.main(String[])
- HelloFOSDEM.main(String[])
- HelloFOSDEM.main(String[])

0: After phase AnalysisGraphBuilder

0 Start
3 LoadField#System.out
5 IsNull
11 If
12 Begin
13 Pi
4 C(Hello, FOSDEM 2025) a java.lang.Object
10 Begin
14 SubstrateMethodCallTarget
6 BytecodeException
21 Invoke!#PrintStream.println
22 Begin
15 ExceptionObject
18 End
17 End
19 Merge
24 Return
20 Phi(6, 15, a java.lang.Object)
26 Unwind

Filters | Bytecode | Control Flow

- 0 getstatic 1
- 3 ldc 13
- 5 invokevirtual 2
- 8 return

SubstrateMethodCallTarget – Properties

Properties

Property	Value
id	14
idx	14
invokeKind	CallTargetNode$Invoke...
targetMethod	java.io.PrintStream.prin...
nodeCostSize	NodeSize.SIZE_0
stamp	void
typeProfile	<null value>
referencedType	<null value>
staticTypeProfile	<null value>
returnStamp	void
nodeSourcePosition	Java: HelloFOSDEM...
methodProfile	<null value>
nodeCostCycles	NodeCycles.CYCLES_0
category	floating
name	SubstrateMethodCallTa...
class	com.oracle.svm.core.n...

SubstrateMethodCallTarget

Stack View

Copyright © 2025, Oracle and/or its affiliates

IGV Graph Phase Example

Outline × | Source Collections

Dump 29628 (Sun Jan 26 12:09:35 CET 2025)

- HelloFOSDEM.main(String[])
  - 0: After phase AnalysisGraphBuilder
  - 1: After phase StaticFinalFieldFolding
  - 2: After phase ImplicitAssertions
  - 3: After phase BoxNodeIdentity
  - 4: After phase PartialEscape
  - 5: After phase Canonicalizer
- HelloFOSDEM.main(String[])
  - 0: InlineBeforeAnalysis after decode
  - 1: After phase Canonicalizer
  - 2: After phase IterativeConditionalElimination
  - 3: After phase BoxNodeIdentity
  - 4: After phase PartialEscape
- HelloFOSDEM.main(String[])
  - 0: After phase AnalysisStrengthenGraphs
- HelloFOSDEM.main(String[])
  - 0: After phase ImplicitAssertions
  - 1: After phase DeadStoreRemoval
  - 2: After phase Canonicalizer
  - 3: After phase Canonicalizer
  - 4: After phase OptimizeExceptionPaths
  - 5: After phase Canonicalizer
  - 6: After phase PhaseSuite
- HelloFOSDEM.main(String[])
  - 0: After phase TrivialInline
  - 1: After phase Canonicalizer
- HelloFOSDEM.main(String[])
  - 0: initial state
  - 1: After phase DeadStoreRemoval
  - 2: After phase RemoveUnwind
  - 3: After phase Canonicalizer
  - 4: After phase DeadCodeElimination
  - 5: After phase IterativeConditionalElimination
  - 6: After phase DominatorBasedGlobalValueNumbe
  - 7: After phase BoxNodeIdentity
  - 8: After phase FinalPartialEscape
  - 9: After phase ReadElimination

Before Analysis

After Substrate World Finalization

Compile

0: After phase AnalysisGraphBuilder ×

0 Start
3 LoadField#System.out
11 If
13 Pi
4 C(Hello, FOSDEM 2025) a java.lang.Object
10 Begin
14 SubstrateMethodCallTarget
21 Invoke!#PrintStream.println
22 Begin
15 ExceptionObject
18 End
17 End
19 Merge
24 Return
20 Phi(6, 15, a java.lang.Object)
26 Unwind

Filters | Bytecode × | Control Flow

- 0 getstatic 1
- 3 ldc 13
- 5 invokevirtual 2
- 8 return

Stack View

SubstrateMethodCallTarget – Properties ×

Properties

Property	Value
id	14
idx	14
invokeKind	CallTargetNode$Invoke...
targetMethod	java.io.PrintStream.prin...
nodeCostSize	NodeSize.SIZE_0
stamp	void
typeProfile	<null value>
referencedType	<null value>
staticTypeProfile	<null value>
returnStamp	void
nodeSourcePosition	Java: HelloFOSDEM...
methodProfile	<null value>
nodeCostCycles	NodeCycles.CYCLES_0
category	floating
name	SubstrateMethodCallTa...
class	com.oracle.svm.core.n...

SubstrateMethodCallTarget

Copyright © 2025, Oracle and/or its affiliates

Compiler-Java Interaction – JVMCI

Compiler needs way to query Java world

- Normal applications do this via reflection
  - But compiler needs additional information about VM internals
  - Also need degree of separation

Solution: (JVMCI) Java-Level JVM Compiler Interface (JEP 243)

- Provides an API for querying information about a JVM instance

“Normal” Reflection Class	Comparable JVMCI Class
java.lang.Class	ResolvedJavaType
java.lang.reflect.Field	ResolvedJavaField
java.lang.reflect.Method	ResolvedJavaMethod

Copyright © 2025, Oracle and/or its affiliates

JVMCI Examples

```
public abstract class CallTargetNode extends ValueNode implements

    @Input protected NodeInputList<ValueNode> arguments;
    protected ResolvedJavaMethod targetMethod;
```


Copyright © 2025, Oracle and/or its affiliates

JVMCI Examples

```
/**
 * The base class of all instructions that access fields.
 */
@NodeInfo(cycles = CYCLES_2, size = SIZE_1)  18 usages  2 inheritors  Thomas Wuerthinger +11
public abstract class AccessFieldNode extends FixedWithNextNode implements Lowerable, OrderedMemoryA

    public static final NodeClass<AccessFieldNode> TYPE = NodeClass.create(AccessFieldNode.class);
    @OptionalInput ValueNode object;  6 usages
    protected final FieldLocationIdentity location;
    protected final ResolvedJavaField field;
    protected final MemoryOrderMode memoryOrder;
```


Copyright © 2025, Oracle and/or its affiliates

Exposing Substrate World through JVMCI

JVMCI serves as interface into Substrate World

Native Image creates own implementation of JVMCI objects

AnalysisType, AnalysisField, and AnalysisMethod
- installed by BytecodeParser

HostedType, HostedField, and HostedMethod
- Created during the “Finalize Substrate World” stage
  - See UniverseBuilder
- Inserted into graphs at the start of Compile stage
  - See AnalysisToHostedGraphTransplanter

Copyright © 2025, Oracle and/or its affiliates

JVMCI implementation benefits

Substrate JVMCI objects wrap Host JVM JVMCI objects

Use Host JVMCI objects to answer many queries

- Method signatures
- Method resolution
- Field properties
- Many more

... But also intercept and alter query results

- Add/Remove Fields
- Change Method Implementations

Copyright © 2025, Oracle and/or its affiliates

JVMCI Object Representation

Graal Graphs refer to JavaConstants

- JVMCI equivalent of Objects

Native Image Builder creates ImageHeapConstant

- Usually backed by Host JVM value

Copyright © 2025, Oracle and/or its affiliates

StrengthenGraphs

Improves graphs based on analysis results

- Injects information into our type system (Stamps)
  - More specific types, nullness information
- Constant folds logic checks (e.g. null, instanceof checks)
- Converts virtual/interface invokes into direct calls

Copyright © 2025, Oracle and/or its affiliates

Goals Revisited

Better Understanding of

1. What phases exist within the builder
2. How to retrieve and view Graal Graphs
3. How the “Substrate Universe” is represented
4. How the “Substrate Universe” relates to the “Java Universe” of the Host JVM

 Copyright © 2025, Oracle and/or its affiliates

Goals Revisited

What phases exist within the builder:

- Analysis
- Finalize Substrate World
- Compile
- Prepare Executable

Copyright © 2025, Oracle and/or its affiliates

Goals Revisited

How to retrieve and view Graal Graphs

Viewable via IGV

- -H:PrintGraph=[Network|File]
- -H:MethodFilter=[methods]
- -H:Dump=[dump level / phases]

 Copyright © 2025, Oracle and/or its affiliates

Goals Revisited

How the “Substrate Universe” is represented

Always interacts via JVMCI interface

First: AnalysisType, AnalysisField, and AnalysisMethod

- installed by BytecodeParser

Later: HostedType, HostedField, and HostedMethod

- Transformed during AnalysisToHostedGraphTransplanter

See HostedUniverse javadoc for more details

Copyright © 2025, Oracle and/or its affiliates

Goals Revisited

How the “Substrate Universe” relates to the “Java Universe” of the Host JVM

Hosted/Analysis JVMCI types wrap Host JVMCI types

- Use Host JVMCI objects for answering many queries
- Modify results to model Substrate World
  - add/remove fields
  - switch out method implementations

 Copyright © 2025, Oracle and/or its affiliates

Thank You

Questions

Copyright © 2025, Oracle and/or its affiliates