Measurement: Infrastructure

Performance Evaluation of Computer Systems

Vojtěch Horký Peter Libič Petr Tůma

Department of Distributed and Dependable Systems
Faculty of Mathematics and Physics
Charles University

2010 – 2021

Outline

1. Java Management Extensions
2. Simple Network Management Protocol
3. Java Virtual Machine Tool Interface

Java Management Extensions (JMX)

A Java framework for management information export and control.

Features include:

- Standard interface to export data collected by instrumentation (MBean).
  - Readable and writable attributes.
  - Operations that can be invoked.
  - Metadata information.
- Standard interface to register available instrumentation (MBean Server).
  - Registry for MBean instances.
  - Unique names and arbitrary properties.
  - Also remote access from outside the JVM.

With JMX we can wrap instrumentation and management functionality in an interface that is standard enough to permit use of generic tools, such as *JConsole* or *VisualVM*.

`https://docs.oracle.com/javase/8/docs/technotes/guides/jmx/index.html`

Standard MBean Interface

```
public interface MyClassMBean {
    public int getState ();
    public void setState (int s);
    public void reset ();
}
```

Interface definition rules:

- Interface name with MBean suffix.
- Getters (and setters) for attributes.
- What is not an attribute accessor is an operation.
- Inheritance (with fairly complex rules) also supported.

Based on code from JMX 1.4 Specification

Standard MBean Implementation

```
public class MyClass implements MyClassMBean {

    private int state = 0;

    public int getState () {
        return (state);
    }
    public void setState (int newState) {
        state = newState;
    }

    public void reset () {
        state = 0;
    }
}
```

Based on code from JMX 1.4 Specification

MBean Server Registration

```
public class Main {
    public static void main (String [] args) {

      MBeanServer mbs =
          ManagementFactory.getPlatformMBeanServer ();
      ObjectName name =
          new ObjectName ("com.example:type=MyClass");

      MyClass mbean = new MyClass ();
      mbs.registerMBean (mbean, name);

      ...
    }
}
```

Based on code from Oracle JMX Tutorial

Standard MBean Notifications

```
public class MyClass
    extends NotificationBroadcasterSupport
    implements MyClassMBean {

    public void setState (int newState) {
        int oldState = state;
        state = newState;

        Notification n =
            new AttributeChangeNotification (
                this, sequenceNumber ++
                System.currentTimeMillis (),
                "State changed", "State", "int",
                oldState, newState);
        sendNotification (n);
```

Based on code from Oracle JMX Tutorial

Dynamic MBean Interface

```
public interface DynamicMBean {
    public Object getAttribute (String attribute);
    public void setAttribute (Attribute attribute);

    public AttributeList getAttributes (
        String [] attributes);
    public AttributeList setAttributes (
        AttributeList attributes);

    public Object invoke (
        String action, Object params [],
        String signature []);

    public MBeanInfo getMBeanInfo ();
}
```

Based on code from JMX 1.4 Specification

Outline

1. Java Management Extensions
2. Simple Network Management Protocol
3. Java Virtual Machine Tool Interface

Simple Network Management Protocol (SNMP)

A standard framework for network device management.

Features include:

- Standard architecture
  - SNMP agents (command responder, notification originator).
  - SNMP managers (command generator, notification receiver).
  - Standard communication protocol.
- Management information model
  - Management information structure.
  - Standard structure bases (MIB).

`https://tools.ietf.org/html/std62`

SNMP Walk Example

```
> snmpwalk -v 1 -c public localhost
...
IF-MIB::ifNumber.0 = INTEGER: 2
IF-MIB::ifIndex.1 = INTEGER: 1
IF-MIB::ifIndex.2 = INTEGER: 2
IF-MIB::ifDescr.1 = STRING: lo
IF-MIB::ifDescr.2 = STRING: wlp3s0
IF-MIB::ifType.1 = INTEGER: softwareLoopback(24)
IF-MIB::ifType.2 = INTEGER: ethernetCsmacd(6)
IF-MIB::ifMtu.1 = INTEGER: 65536
IF-MIB::ifMtu.2 = INTEGER: 1500
IF-MIB::ifPhysAddress.1 = STRING:
IF-MIB::ifPhysAddress.2 = STRING: 00:11:22:33:44:55
IF-MIB::ifInOctets.1 = Counter32: 4448447
IF-MIB::ifInOctets.2 = Counter32: 179122682
IF-MIB::ifInUcastPkts.1 = Counter32: 30093
IF-MIB::ifInUcastPkts.2 = Counter32: 267652
IF-MIB::ifOutOctets.1 = Counter32: 4448447
IF-MIB::ifOutOctets.2 = Counter32: 356107468
IF-MIB::ifOutUcastPkts.1 = Counter32: 30093
IF-MIB::ifOutUcastPkts.2 = Counter32: 520460
```

SNMP Object Information Example

```
> snmptranslate -Td IF-MIB::ifInOctets
IF-MIB::ifInOctets
ifInOctets OBJECT-TYPE
  -- FROM IF-MIB
  SYNTAX  Counter32
  MAX-ACCESS  read-only
  STATUS  current
  DESCRIPTION "The total number of octets received on the interface,
              including framing characters.

              Discontinuities in the value of this counter can occur at
              re-initialization of the management system, and at other
              times as indicated by the value of
              ifCounterDiscontinuityTime."
::= { iso(1) org(3) dod(6) internet(1) mgmt(2) mib-2(1) interfaces(2) ifTabl
```

Outline

1. Java Management Extensions
2. Simple Network Management Protocol
3. Java Virtual Machine Tool Interface

Java Virtual Machine Tool Interface (JVMTI)

A C and C++ interface for attaching native tools to the Java Virtual Machine.

Features include:

- Delivering event notifications through callbacks.
- Providing functions to query and modify JVM state.
- Defining argument passing rules (JNI).
- Defining JVM execution phases.

With JVMTI we create *agents*, shared libraries that can be loaded into the JVM using the `agentlib` or `agentpath` command line arguments.

`https://docs.oracle.com/javase/8/docs/platform/jvmti/jvmti.html`
`https://docs.oracle.com/javase/8/docs/technotes/guides/jvmti`

JVMTI Execution Phases

JVM execution phases describe what state the JVM is in:

JVMTI_PHASE_ONLOAD Inside `Agent_OnLoad` before VM start

JVMTI_PHASE_PRIMORDIAL After `Agent_OnLoad` but before VM start

JVMTI_PHASE_START After VM start but before VM init

JVMTI_PHASE_LIVE After VM init but before VM death

JVMTI_PHASE_DEAD After VM death or after init failure

Most JVMTI functions only work in live phase.

JVMTI Defined Events

JVMTI defines and reports important JVM execution events:

- Breakpoint
- Class file load hook, class load and prepare
- Compiled method load and unload
- Exception throw and catch
- Field access and modification (watched fields)
- Garbage collection start and end
- Method entry and exit (all executed methods)
- Monitor wait and waited, contended enter and entered
- Thread start and end
- VM start and initialization and death

Some of these events are better intercepted
through bytecode instrumentation.

JVMTI Functions

JVMTI provides many functions to access and manipulate JVM state:

JVMTI memory management agent memory management, correct deallocation particularly important

Thread functions thread information, suspend and resume and stop, start agent thread, local storage

Stack frame get stack trace, pop frame (return before invoke), force early method return

Heap visiting all or reachable objects, object tagging, forcing GC

Class class and classloader info, methods, fields, retransform classes, redefine classes

Object size, hash, monitor usage

Fields and methods get information, watch field

Timers Java managed timer data

Creating Agents with JVMTI

An agent is a shared library that implements
the `Agent_OnLoad` and `Agent_OnUnLoad` functions.

`Agent_OnLoad (JavaVM *vm, char *opts, void *rsvrd)`

Agent entry point, called after invocation from command line and before starting the VM. Used to request JVMTI capabilities and initialize event callbacks.

JVMTI function call pattern:

```
C++: jvmti_env->Function (parameters);
C:   (*jvmti_env)->Function (jvmti_env, parameters);
```

Agent_OnLoad Example

```
JNIEXPORT jint JNICALL
Agent_OnLoad (JavaVM *vm, char *options, void *reserved) {
    // Errors in example silently ignored.
    jvmtiEnv *jvmti;
    vm->GetEnv ((void**) (&jvmti), JVMTI_VERSION);

    jvmtiEventCallbacks callbacks;
    memset (&callbacks, 0, sizeof (callbacks));
    callbacks.VMInit = &on_vm_init;
    callbacks.ThreadStart = &on_thread_start;
    jvmti->SetEventCallbacks (&callbacks, sizeof (callbacks));

    jvmti->SetEventNotificationMode (JVMTI_ENABLE,
        JVMTI_EVENT_VM_INIT, (jthread) NULL);
    jvmti->SetEventNotificationMode (JVMTI_ENABLE,
        JVMTI_EVENT_THREAD_START, (jthread) NULL);

    return (JNI_OK);
}
```

VMInit Callback Example

```
static void JNICALL
on_vm_init (jvmtiEnv *jvmti, JNIEnv *env, jthread thread) {
    // Errors in example silently ignored.
    jclass thread_class = env->FindClass ("java/lang/Thread");

    jmethodID thread_constructor = env->GetMethodID (
        thread_class, "<init>", "()V");

    jthread thread_object = env->NewObject (
        thread_class, thread_constructor);

    jvmti->RunAgentThread (
        thread_object, &my_thread_main,
        NULL, JVMTI_THREAD_NORM_PRIORITY);
}

static void JNICALL
my_thread_main (jvmtiEnv* jvmti_env, JNIEnv* env, void *unused) {
    ...
}
```

ThreadStart Callback Example

```
// Thread local variables work fine.
static __thread jlong my_thread_id;

static void JNICALL
on_thread_start (jvmtiEnv *jvmti, JNIEnv *env, jthread thr) {
    // Errors in example silently ignored.
    jclass thread_class = env->FindClass ("java/lang/Thread");

    jmethodID method_getId = env->GetMethodID (
        thread_class, "getId", "()J");

    my_thread_id = env->CallLongMethod (thr, method_getId);
}
```

Heap Iteration Example

```
static jint JNICALL
object_counter_callback (
    jlong class_tag, jlong size, jlong* tag_ptr,
    jint length, void *user_data) {
    long *counter = (long *) user_data;
    (*counter) ++;
    return (JVMTI_VISIT_OBJECTS);
}

static void JNICALL
on_vm_init (jvmtiEnv *jvmti, JNIEnv *env, jthread thread) {
    // Count objects on heap.
    jvmtiHeapCallbacks callbacks;
    memset (&callbacks, 0, sizeof (callbacks));
    callbacks.heap_iteration_callback = &object_counter_callback;
    long counter = 0;
    jvmtiError err = (*jvmti)->IterateThroughHeap (
        jvmti, 0, NULL, &callbacks, &counter);
    ...
}
```