Java Reflexion, Java Instrumentation and JMX

Vania Marangozova-Martin
Associate Professor, University of Grenoble – LIG, France

Vania.Marangozova-Martin@imag.fr

Web Site: http://acs.forge.imag.fr

Administrivia

	Week	Monday 8:15 –9:45 (1h30)	Tuesday, 08:15 – 9:45 (1h30)	Thursday, 08:15 – 9:45 (1h30)
5	01-févr		Introduction	Java Instrumentation
6	08-févr		Tutorial JMX	JMX
7	15-févr		Design Patterns	JMX : demos
8	22-févr	Interruption week		
9	29-févr		Tutorial Design Patterns	Design Patterns
10	07-mars	No ACS (Vania away)		
11	14-mars	Design Patterns	Design Patterns	Design Patterns demos
12	21-mars	NF Properties and AOP	AOP Tutorial	AOP Logging Lab
13	28-mars		Article Presentation	MOP
14	04-avr	Article Presentation	Article Presentation	MOP
15	11-avr	MOP tutorial	Article Presentation	MOP
16	18-avr	Interruption week		
17	25-avr		No ACS	MOP demos
	02-mai	EXAMS		

Java Reflexion

- Java reflexion is the mechanism provided by the JVM to obtain information about the classes, interfaces, methods and fields at runtime
- Using this information it is possible
  - to call methods
  - access fields
  - implement interfaces
  - …

Lab 1: The preliminary question

- Have a basic example to test reflection on

```
public class Person {

   private String firstName;
   private String familyName;
   private int age;
   private int SSnumber;
   private int telNumber;

   public Person(int ss, String firstName, String secondName) {
      SSnumber = ss;
      this.firstName = firstName;
      this.familyName = secondName;
   }
      …
```

Person.java cont.

```
public class Person {

    public int getTelNumber() {return telNumber;}
    public void setTelNumber(int tel) {telNumber = tel;}
    public String getFirstName() {return firstName;}
    public void setFirstName(String first) {firstName = first;}
    public String getFamilyName() {return familyName;}
    public void setFamilyName(String f) {familyName = f;}
    public int getAge() {return age;}
    public void setAge(int age) {this.age = age;}
    public int getSSnumber() {return Ssnumber;}
    public void setSSnumber(int sSnumber) {SSnumber = sSnumber;}
    public String toString() {
        String res = "["+ SSnumber +
                     ":" + firstName +
                     ":" + familyName +
                     ":" + age +"]";
        return res;
    }
```


Catalogue.java

```
public class Catalogue {

    private static Hashtable<Integer, Person> catalogue
        = new Hashtable<Integer, Person>();

    public static void add(Person p) {
        if (catalogue.get(p.getSSnumber()) == null)
            catalogue.put(p.getSSnumber(), p);
    }

    public static void delete(int ss) {
        for (Enumeration<Integer>  k =  catalogue.keys();
                k.hasMoreElements();) {
            Integer kss = k.nextElement();
            if (kss.intValue() == ss) catalogue.remove(kss);
        }
    }
…
```


Catalogie.java cont.

```
    public static List<Integer> find(String firstName,
                                     String familyName) {
        …
    }

    public static void update(int ss, int telNumber) {
        …
    }

    public static void list() {
        …
    }

    public static Person findPerson(int ss) {
        return catalogue.get(ss);
    }
```


TestCatalogie.java

```
public class TestCatalogue {

    public static void main(String[] args) {
        Person p1 = new Person(1, "Vania", "Marangozova"); p1.setAge(25);
        Person p2 = new Person(2, "Michael", "Jackson");p2.setAge(35);
        Person p3 = new Person(3, "Leslie", "Lamport");p3.setAge(55);

        Catalogue.add(p1);
        Catalogue.add(p2);
        Catalogue.add(p3);

        Catalogue.list();

        List<Integer> l = Catalogue.find("Vania", "Marangozova");
        for(int ss : l) System.out.println(Catalogue.findPerson(ss));
    }
}
```

Java Reflection: Getting the class information

```
try {
   //getting the class
   Class<?> c = Class.forName("acs.instr.test.Person");
```


Java Reflection: Getting the fields

```
Field[] fs = c.getFields();
for (Field f : fs) {
      out.format("  %s%n", f.toGenericString());
}
```

- What hapens with private fields/methods?
- You have at your disposal, for methods and fields
  - `setAccessible(true)`

Java Reflection: Getting the methods

```
Method[] ms = c.getMethods();
for (Method m : ms) {
   out.format("  Method: %s%n", m.toGenericString());
}
```


Java Reflection: Getting and using the constructors

```
Constructor<?>[] cs = c.getConstructors();
for (Constructor<?> co : cs) {
   out.format("  Method: %s%n", co.toGenericString());
}

//invoking a constructor
Constructor<?> cp =
c.getConstructor(new Class[]{ int.class,
                              String.class,
                              String.class});
Person p_refl =
   (Person)cp.newInstance(4, "Vania", "Marangozova");
out.println(p_refl);
```

Java Reflection: Getting and using the methods

```
Method mp =
    p_refl.getClass().getMethod("setAge",
                                new Class[]{int.class});
mp.invoke(p_refl, 15);
out.println(p_refl);
```


And another tutorial…

- http://tutorials.jenkov.com/java-reflection/index.html

Java Instrumentation

java.lang.instrument

Interface Instrumentation

`public interface Instrumentation`

This class provides services needed to instrument Java programming languag[illegible] byte-codes to methods for the purpose of gathering data to be utilized by tools. Since the changes are purely additive, these tools do not modify application state or behavior. Examples of such benign tools include monitoring agents, profilers, coverage analyzers, and event loggers.

There are two ways to obtain an instance of the `Instrumentation` interface:

1. When a JVM is launched in a way that indicates an agent class. In that case an `Instrumentation` instance is passed to the `premain` method of the agent class.
2. When a JVM provides a mechanism to start agents sometime after the JVM is launched. In that case an `Instrumentation` instance is passed to the `agentmain` method of the agent code.

These mechanisms are described in the package specification.

Once an agent acquires an `Instrumentation` instance, the agent may call methods on the instance at any time.

Since:

Defining an agent…

```
package acs.instr.test;

import java.lang.instrument.Instrumentation;

public class TestJavaInstrumentation {


   public static void premain(String args, Instrumentation inst) {
       Person obj = new Person(1, "Vania", "Marangozova");
       long size = inst.getObjectSize(obj);
       System.out.println("Bytes used by object: " + size);
   }

}
```

Tutorial at: http://www.javamex.com/tutorials/memory/instrumentation.shtml

Package the agent into a jar

- manifest.txt file
  - `Premain-Class: mypackage.MyAgent`
- compile
  - `jar -cmf manifest.txt agent.jar classes/acs/instr/test/ TestJavaInstrumentation.class`

Launch

- `java –javaagent:agent.jar -cp classes acs.instr.test.TestCatalogue`

The Instrumentation interface…

- is actually meant to be used with class transformers i.e classes that transform other classes
- The tutorial at `http://blog.javabenchmark.org/2013/05/java-instrumentation-tutorial.html` explains that

You have to…

- Install javassist: a tool for bytecode manipulation
  - API:

`http://jboss-javassist.github.io/javassist/html/`

  - Download:

`http://jboss-javassist.github.io/javassist/`

There is the Sleeping tutorial…

- Example of basic profiling

1. Write a trivial class that will be instrumented
2. Write a ClassFileTransformer to inject some code to print method execution time
3. Write an agent that registers the previous transformer
4. Write a test

The class to be instrumented

```
package org.javabenchmark.instrumentation;

public class Sleeping {

    public void randomSleep() throws InterruptedException {

        // randomly sleeps between 500ms and 1200s
        long randomSleepDuration =
             (long) (500 + Math.random() * 700);

        System.out.printf("Sleeping for %d ms ..\n",
             randomSleepDuration);

        Thread.sleep(randomSleepDuration);
    }
}
```


The test

```
public class TestSleepingTransformer {

       public static void main(String[] args) {
       System.out.println("Program started...");
       Sleeping ist = new Sleeping();
       try {
           ist.randomSleep();
           System.out.println("Calling sum...");
           System.out.println(ist.sum(3,3));
       } catch (Exception e) {
           e.printStackTrace();
       }

   }

}
```


The transformer class

```
public class SleepingClassTransformer implements ClassFileTransformer {

   public byte[] transform(…) throws IllegalClassFormatException {
      …
     if (className.equals("acs/instr/TestSleepingTransformer")) {

     try {
       ClassPool cp = ClassPool.getDefault();
       CtClass cc = cp.get("acs.instr.Sleeping");

       System.out.println("Modifying randomSleep method...");
       CtMethod  m = cc.getDeclaredMethod("randomSleep");
       m.addLocalVariable("elapsedTime", CtClass.longType);
       m.insertBefore("elapsedTime = System.currentTimeMillis();");
      m.insertAfter("{elapsedTime = System.currentTimeMillis() -
                    elapsedTime;"+
                     "System.out.println(\"Method Executed in ms: \"
                     + elapsedTime);}");
…
```

Executing…

```
>java -javaagent:instr2.jar
       -cp classes:javassist.jar
       acs.instr.TestSleepingTransformer
Modifying randomSleep method...
Program started...
Sleeping for 922 ms ..
Method Executed in ms: 939
```


Your goal

1. Add a new method to the Sleeping class
2. Quit the program when the sum method is called
   1. Use either method.insert..
   2. or ExprEditor
3. Count the number fo times each method is called and print the statistics

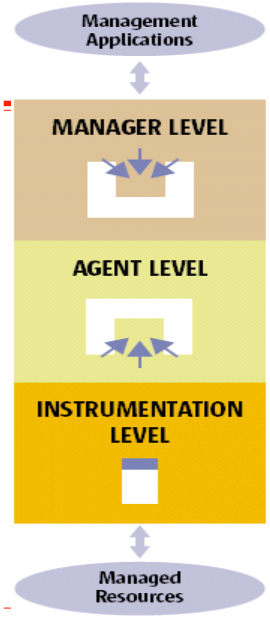
The General Picture

- Reflection: get information
- Instrumentation: modify code so as to plug "utility" code
  - profiling
  - monitoring
  - reaction to erroneous behavior
  - … management
- JMX generalizes the approach

JMX = Java Management Extensions

- "*The JMX specification defines*
  - *an architecture,*
  - *the design patterns,*
  - *the APIs,*
  - *and the services for application and network management and monitoring*
  - *in the Java programming language*."

http://docs.oracle.com/javase/1.5.0/docs/guide/jmx/overview/intro.html#wp5529



JMX Type	Developer Type	Benefits
JMX Manager	Developers of management solutions	Quick and simple integration with existing manage-ment infrastrusture
JMX Agent	Developers of management solutions	Dynamic scalability by plugging in agent services on the fly
JMX Resource	All Java developers	Standard man-ageability for Java technology across all industries

JMX Instrumentation

- *Resources,*
- *such as applications, devices, or services,*
- *are instrumented using Java objects called Managed Beans (MBeans).*
- *MBeans expose their management interfaces, composed of attributes and operations, through a JMX agent for remote management and monitoring.*

http://www.oracle.com/technetwork/articles/javase/jmx-138825.html

MBeans

- An MBean is a Java object that implements a specific interface.
- The management interface of an MBean is represented as:

(1) valued attributes that can be accessed;
(2) operations that can be invoked;
(3) notifications that can be emitted; and
(4) the constructors.

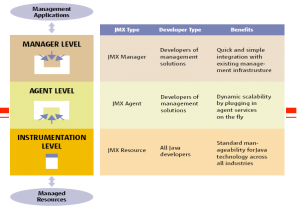
Example

```
package tutorial;

public interface HelloMBean {

        public void setMessage(String message);

        public String getMessage();

        public void sayHello();
    }
```

Example

```
public class Hello {

    private String message = null;

    public Hello() {message = "Hello there";}

    public Hello(String message) {this.message = message;}

    public void setMessage(String message) {this.message = message;}

    public String getMessage() {return message;}

    public void sayHello() {System.out.println(message);}

}
```


The agent level



- This tier contains the
- JMX agents used to expose the MBeans.
- It provides a specification for implementing agents, which control the resources and make them available to remote management applications.
- Agents are usually located on the same machine as the resources they manage, but this is not a requirement.

The agent level

- The JMX agent consists of an MBean server and a set of services for handling Mbeans.
  - **MBean Server**: A registry of objects that are exposed to management operations in an agent.
  - **Agent Services**: Objects that can perform management operations on the MBeans registered in the MBean server.

Back to the example

- An implemntation of an agent to
  - Get the platform MBeanServer
  - Register an instance of the Hello MBean

Example

```
public class SimpleAgent {
    private MBeanServer mbs = null;

    public SimpleAgent() {

        // Get the platform MBeanServer
        mbs = ManagementFactory.getPlatformMBeanServer();

        // Unique identification of MBeans
        Hello helloBean = new Hello();
        ObjectName helloName = null;

        try {
            helloName = new ObjectName("SimpleAgent:name=hellothere");
            mbs.registerMBean(helloBean, helloName);
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
```


Example

```
// Utility method: so that the application continues to run
    private static void waitForEnterPressed() {
        try {
            System.out.println("Press  to continue...");
            System.in.read();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String argv[]) {
        SimpleAgent agent = new SimpleAgent();
        System.out.println("SimpleAgent is running...");
        SimpleAgent.waitForEnterPressed();
    }
```


Management Level

- *"This tier contains the components that enable management applications to communicate with JMX agents."*
- Management tools include `jconsole`

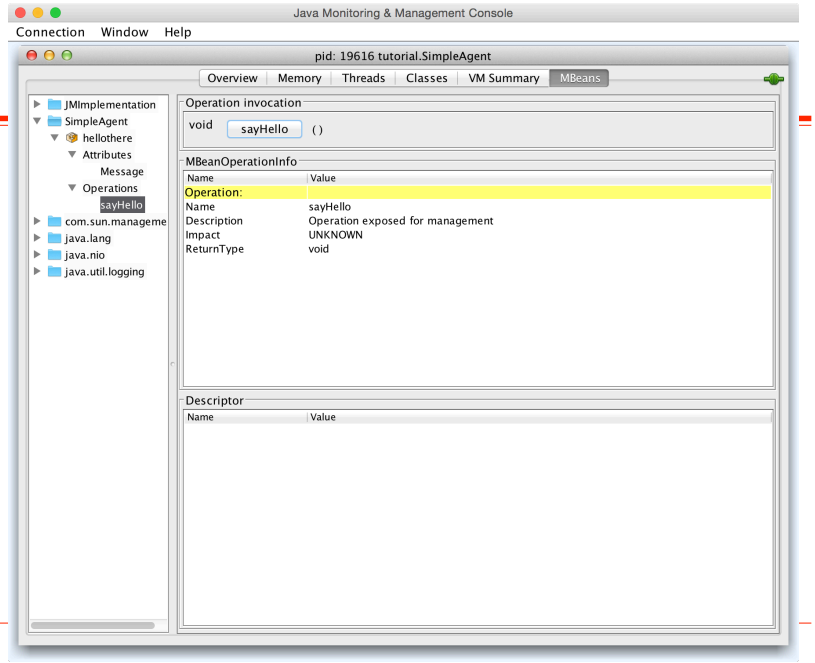
Running the example…

- `1 terminal`

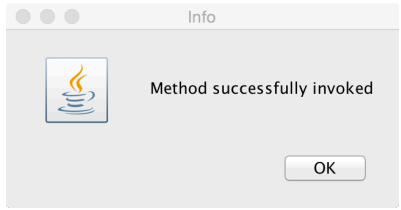
```
>java -
Dcom.sun.management.jmxremote
tutorial.SimpleAgent
```

- `another terminal`

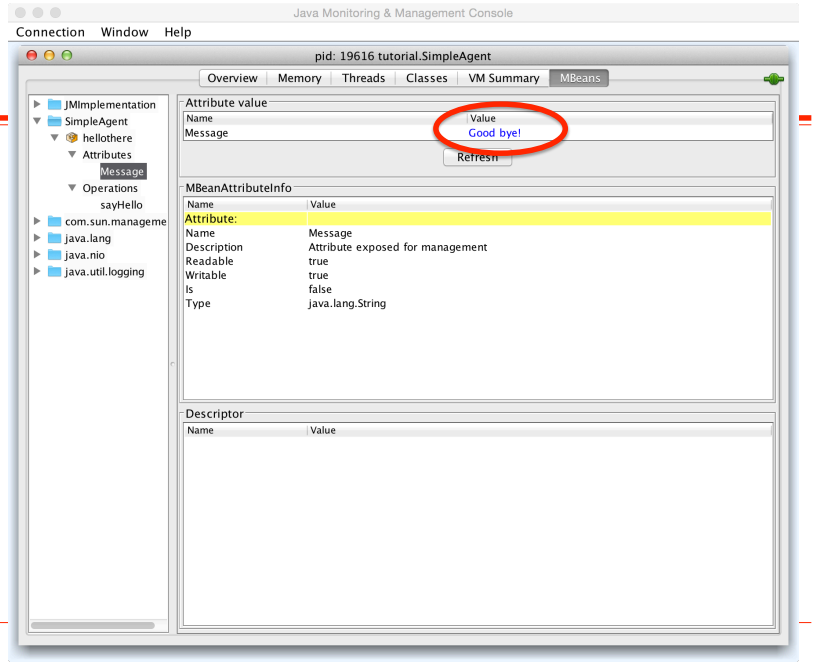
```
>jconsole
```



Invoking a method…




```
[~/Code/workspaceJava/JMX_2016/bin] java -
Dcom.sun.management.jmxremote tutorial.SimpleAgent
SimpleAgent is running...
Press  to continue...
Hello there
```

Invoking sayHello() again…

```
[~/Code/workspaceJava/JMX_2016/bin] java -
Dcom.sun.management.jmxremote tutorial.SimpleAgent
SimpleAgent is running...
Press  to continue...
Hello there
Good bye!
```

SimpleStandard Example

1. Dynamic Mbeans
2. Remote Bean management using rmiregistry

The Interface

```
public interface SimpleStandardMBean {

    public String getState();
    public void setState(String s);

    public int getNbChanges();
    public void reset();
}
```


The Standard Bean

NAMING CONVENTION

```
public class SimpleStandard
    extends NotificationBroadcasterSupport
    implements SimpleStandardMBean {

    public String getState() {return state;}
    public void setState(String s) {
        state = s;
        nbChanges++;
    }

    public int getNbChanges() {return nbChanges;}
    public void reset() {
        AttributeChangeNotification acn =
            new AttributeChangeNotification(this,
                0, 0,
                "NbChanges reset","NbChanges","Integer",
                new Integer(nbChanges),
                new Integer(0));
        state = "initial state";nbChanges = 0;nbResets++;
        sendNotification(acn);
    }
```


The DynamicBean

NAMING CONVENTION

```
public class SimpleDynamic
    extends NotificationBroadcasterSupport
    implements DynamicMBean {


    public SimpleDynamic() {buildDynamicMBeanInfo();}

    public Object getAttribute(String attribute_name)
        throws AttributeNotFoundException,
               MBeanException,
               ReflectionException {
        if (attribute_name == null) {
            throw …
        if (attribute_name.equals("State")) {
            return getState();}
        if (attribute_name.equals("NbChanges")) {
            return getNbChanges();
        }
       …
    }
```

The Server

```
public class Server {
 public static void main(String[] args) {
    try {
        // Instantiate the MBean server
        MBeanServer mbs = MBeanServerFactory.createMBeanServer();

        String mbeanClassName = "Basic.SimpleStandard";
        String mbeanObjectNameStr =
        domain + ":type=" + mbeanClassName + ",index=1";
        ObjectName mbeanObjectName =
        createSimpleMBean(mbs, mbeanClassName, mbeanObjectNameStr);

        manageSimpleMBean(mbs, mbeanObjectName, mbeanClassName);

        mbeanClassName = "Basic.SimpleDynamic";
        mbeanObjectNameStr =
        domain + ":type=" + mbeanClassName + ",index=1";
        mbeanObjectName =
        createSimpleMBean(mbs, mbeanClassName, mbeanObjectNameStr);

        manageSimpleMBean(mbs, mbeanObjectName, mbeanClassName);
```


The Server cont.

```
public class Server {
 …
    // Create an RMI connector server
            //
    JMXServiceURL url = new JMXServiceURL(
             "service:jmx:rmi:///jndi/rmi://localhost:9999/server");
    JMXConnectorServer cs =
        JMXConnectorServerFactory.newJMXConnectorServer(url, null, mbs);

    // Start the RMI connector server
    cs.start();
```


The Client

```
public class Client {

  public static void main(String[] args) {
    try {
    // Create an RMI connector client and connect it to the RMI connector server
    JMXServiceURL url = new JMXServiceURL(
    "service:jmx:rmi:///jndi/rmi://localhost:9999/server");
    JMXConnector jmxc = JMXConnectorFactory.connect(url, null);

    // Get an MBeanServerConnection
    MBeanServerConnection mbsc = jmxc.getMBeanServerConnection();

    ObjectName stdMBeanName =
        new ObjectName(domain +":type=Basic.SimpleStandard,index=2");
    mbsc.createMBean("Basic.SimpleStandard", stdMBeanName, null, null);

    …

    mbsc.setAttribute(stdMBeanName, new Attribute("State", "changed state"));

    …
```


References

- http://docs.oracle.com/javase/1.5.0/docs/guide/jmx/
- http://docs.oracle.com/javase/7/docs/technotes/guides/management/overview.html
- http://docs.oracle.com/javase/7/docs/technotes/guides/management/jconsole.html
- http://docs.oracle.com/javase/1.5.0/docs/guide/jmx/examples.html
- http://www.oracle.com/technetwork/articles/javase/jmx-138825.html