Java Management Extensions (JMX) Circa 2008

Dustin Marx
Raytheon Company
About This Presentation

- Focus is recent JMX developments and features
  - Not an overall JMX introduction
- JMX basics will be briefly reviewed
  - Understanding JMX basics helps appreciate new JMX features better
  - Cover basics related to new JMX features
  - No previous JMX experience required!
- The “NEW” icon marks slides added to this presentation since original CD version
Why JMX?

- "Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning."
  - Rich Cook

- JMX helps figure out what has happened and what is happening
History of JMX: JSRs

- JSR 3 – JMX Specification
- JSR 77 – J2EE Management
- JSR 88 – Java EE Application Deployment
- JSR 160 – Remote JMX
- JSR 174 – Monitoring & Management Specification for JVM
- JSR 262 – Web Services Connector for JMX
- Withdrawn JMX-related JSRs: 70, 71, 146
# Recent Java SE JMX Features

<table>
<thead>
<tr>
<th></th>
<th>J2SE 5 (Sun)</th>
<th>Java SE 6 (Sun)</th>
<th>Java SE 7* (Sun)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>JMX Version</strong></td>
<td>JMX 1.2</td>
<td>JMX (w/Remote) 1.4 (RI)</td>
<td>JMX 2 (JSR-255)</td>
</tr>
<tr>
<td><strong>Built-in Tools</strong></td>
<td>JConsole</td>
<td>JConsole</td>
<td>JConsole</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VisualVM(6u7)</td>
<td>VisualVM</td>
</tr>
<tr>
<td><strong>Other Notables</strong></td>
<td>MXBeans</td>
<td>MXBeans</td>
<td>Web Services Connector (JSR-262)</td>
</tr>
<tr>
<td></td>
<td>AttachAPI</td>
<td>AttachAPI</td>
<td>Annotations</td>
</tr>
<tr>
<td></td>
<td>Generics</td>
<td>Generics</td>
<td></td>
</tr>
</tbody>
</table>

* Java SE 7 statements are forward-looking and subject to change.

6u7 VisualVM is included in Sun’s Java SE implementation as of Java SE 6 Update 7.
Platform JMX Server

- A JMX Server was delivered as part of the JVM starting with J2SE 5
- Platform MBean server allows custom MBeans to be hosted without need for third-party JMX implementation
- J2SE 5 took advantage of platform JMX server to expose JVM monitoring and management capabilities
- JConsole unofficially included in J2SE 5
Java SE 6
JMX Enhancements

- Improved JConsole
  - JConsole Custom Plug-ins
- Attach API
- MXBean (Customizable)
- Descriptor Support Extended to All MBean Types

- New JMX Support Classes
- Generics Support
- Improved JMX Monitors
- Simplified Open MBeans Development
- Advanced Sample
JConsole

- JConsole first delivered with J2SE 5
- JConsole is a graphical client for basic management and monitoring capabilities
- Supports reading/writing JVM built-in MBeans
- Supports reading/writing custom MBeans
  ➢ Application server and third-party management interfaces also supported
Java SE 6
JConsole Improvements

- Now “officially supported”
- Improved look and feel
  - Windows and GNOME
- Automatically detects JMX-managed applications – Attach API
- Custom plug-in support (custom tab)
- More information and improved presentation
JConsole Plug-in API: Custom Tab

- Create custom JConsole tabs in addition to six default tabs
- JConsole custom plug-in approach uses Java SE 6 features:
  - SwingWorker
  - Pluggable Service Providers
- JConsole comes with the JTop example custom plug-in
- Use JConsole custom tabs in VisualVM
Using Custom JConsole Tabs

- Adding custom JConsole tab is simple
  - Extend JConsole Plugin
    - Implement `getTabs` and `newSwingWorker` methods
  - Create file exactly named entry `META-INF/services/com.sun.tools.jconsole.JConsolePlugin`
    - Contains fully qualified names of custom plug-in tab classes (uses services provider loading)
  - Package custom plugin class and `META-INF` file in single JAR
  - Run JConsole with plugin:
    - `jconsole -pluginpath <<path_to_plugin>>`
Attach API

- Attach tools to another, already running JVM
- Allows Java SE 6 JConsole to detect running managed applications
  - No more need for J2SE 5-required system property `-Dcom.sun.management.jmxremote`
  - Only works for local processes running under same user
- Sun extension `(com.sun.tools.attach)`
MBeans: Managed Beans

- Represents and exposes something to be monitored and managed

Five types of MBeans:

- Standard
- MXBeans (Java 5/6)

- Dynamic
- Model
- Open
Standard MBeans

- Based on naming convention (introspection)
  - Standard MBean implementation class implements interface of same name with "MBean" appended to end
  - JMX Developer writes the implementation class and its interface
- Simplest of original MBean types
- Recommended JMX Best Practice
  - Interfaces enable client proxies
Standard MBean Example: Interface

```java
public interface CalculatorMBean {
    public int getLastRemainder();

    public int add(final int augend, final int addend);

    public int subtract(final int minuend, final int subtrahend);

    public int multiply(final int factor1, final int factor2);

    public double divide(final int dividend, final int divisor);
}
```
Standard MBean Example: Implementation Class

```java
public class Calculator implements CalculatorMBean {
    private int lastRemainder = 0;

    public int getLastRemainder() { return lastRemainder; }

    public int add(final int augend, final int addend) {
        return augend + addend;
    }

    public int subtract(final int minuend, final int subtrahend) {
        return minuend - subtrahend;
    }

    public int multiply(final int factor1, final int factor2) {
        return factor1 * factor2;
    }

    public double divide(final int dividend, final int divisor) {
        lastRemainder = dividend % divisor;
        return dividend / divisor;
    }
}
```
Dynamic MBeans

- No introspection or naming conventions
  - Metadata is explicitly provided at runtime
- More work to code than Standard MBeans
- Unnecessary when management interface is static
- Implement interface `javax.management.DynamicMBean`
Model MBeans

- “Wrap” existing resource that has no JMX awareness
  - Bridge between JMX management and non-JMX resource
- Popular for frameworks (Spring)
- Specialization of (extends) Dynamic MBean
- JMX implementations must provide Model MBean implementation (RequiredModelMBean)
- Richness of Dynamic MBean to easily wrap non-JMX resources
Model MBean Example

- Hand-written Model MBeans tend to be extremely verbose
- Single Calculator-based example will take several slides to cover
- Benefits include
  - Significantly greater detail for generic JMX client
  - Ability to relatively easily (if tediously) “wrap” non-JMX resource
Model MBean Example:
1 – Non-JMX Calculator Class

```java
public class Calculator {
    private int lastRemainder = 0;
    public int getLastRemainder()
    { return this.lastRemainder; }
    public int add(final int augend, final int addend)
    { return augend + addend; }
    public int subtract(final int minuend,
                        final int subtrahend)
    { return minuend - subtrahend; }
    public int multiply(final int factor1,
                        final int factor2)
    { return factor1 * factor2; }
    public double divide(final int dividend,
                         final int divisor)
    { lastRemainder = dividend % divisor;
      return dividend / divisor; }
}
```
Model MBean Example:
2 – Create Model MBean

```java
final ModelMBeanInfoSupport modelMBeanInfo =
    new ModelMBeanInfoSupport(
        Calculator.class.getName(),
        "A simple integer calculator.",
        buildModelMBeanAttributeInfo(),
        null,           // constructors
        buildModelMBeanOperationInfo(),
        null,           // notifications
        buildDescriptor() );
final RequiredModelMBean modelmbean =
    new RequiredModelMBean(modelMBeanInfo);
modelmbean.setManagedResource(
    new Calculator(), "ObjectReference");
```
Model MBean Example: 3 – Constructing Descriptor

```java
private Descriptor buildDescriptor()
{
    final Descriptor descriptor = new DescriptorSupport();
descriptor.setField("name", "ModelMBeanInTheRaw");
descriptor.setField("descriptorType", "mbean");
return descriptor;
}
```

**Note:** Before Java SE 6, only Model MBeans could have Descriptors. Now, all MBeans can have Descriptors.
Model MBean Example: 4 – Build Operations Info

Build Parameters to Operations First

```java
final MBeanParameterInfo augendParameter =
   new MBeanParameterInfo(
       "augend",
       Integer.TYPE.toString(),
       "The first parameter in the addition (augend)."
   );
```

Add Parameters to Operations

```java
final ModelMBeanOperationInfo addOperationInfo =
   new ModelMBeanOperationInfo( "add",
       "Integer Addition",
       new MBeanParameterInfo[]
          {augendParameter, addendParameter},
       Integer.TYPE.toString(),
       ModelMBeanOperationInfo.INFO );
```
Model MBean Example: 5 – Hard-coded Attribute Info

One method for specifying an attribute’s metadata is with custom-coding of the metadata as shown here.

```java
// Explicitly spell out attribute metadata
final ModelMBeanAttributeInfo lastRemainderAttribute
    = new ModelMBeanAttributeInfo(
            "lastRemainder", // name
            Integer.TYPE.toString(), // type
            "Remainder that would have resulted from last " + "integer division", // description
            true, // readable
            false, // not writable
            false ); // no "is" getter
```
Model MBean Example: 6 – Introspected Attr Info

Can use introspection to populate attribute metadata

```java
final ModelMBeanAttributeInfo lastOperationTypeAttribute = new ModelMBeanAttributeInfo("lastOperationType",
    "Last calculator operation invoked.",
    Calculator.class.getMethod(attrGetName),
    null);
```

Build array of all attributes’ metadata info instances

```java
return new ModelMBeanAttributeInfo[]
    {lastRemainderAttribute, lastOperationTypeAttribute};
```
Generic client (like JConsole) can present metadata

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>augend</td>
<td>The first parameter in the addition (augend)</td>
<td>int</td>
</tr>
<tr>
<td>addend</td>
<td>The second parameter in the addition (addend)</td>
<td>int</td>
</tr>
</tbody>
</table>

```
int add (augend 0, addend 0)
```
JConsole without Metadata

Introspection cannot supply names of operation parameters
Model MBean Example

Summarized

- Only portions of the metadata coding needed for Model MBean shown
- Advantages are extremely descriptive information for JMX clients and ability to wrap non-JMX classes
- There are easier ways to employ Model MBeans that will be covered later
  - Spring, Easy MBean, Commons Modeler
Open MBean

- “Open” to generic and non-Java clients
  - Exposed data types constrained to predefined data types
  - JMX support required since JMX 1.2
- Specialization of (extends) Dynamic MBean
- Combines advantages of Dynamic MBean descriptive richness with generic client support
- Designed for greatest client interoperability
MXBeans

- Platform MXBeans introduced in J2SE 5
- Customizable MXBeans introduced in Java SE 6
- Constrains types of data that can be returned (Open Types)
- Combines simplicity of Standard MBeans with general client interoperability of Open MBeans
More on MXBeans

- Similarities to Standard MBeans
  - May adhere to naming convention of implementation class implementing interface with same name + MXBean suffix
  - Can use annotation `@MXBean` for interface instead of interface naming convention
  - Implementation class need only implement MXBean interface
    - No longer needs to be named similarly
    - Same package restriction removed as well
public interface CalculatorMXBean
{
    public int getLastRemainder();

    public int add(final int augend, final int addend);

    public int subtract(final int minuend, final int subtrahend);

    public int multiply(final int factor1, final int factor2);

    public double divide(final int dividend, final int divisor);
}
MXBeans Example: Implementation

```
public class Calculator implements CalculatorMXBean {
    private int lastRemainder = 0;

    public int getLastRemainder() { return lastRemainder; }

    public int add(final int augend, final int addend) {
        return augend + addend;
    }

    public int subtract(final int minuend, final int subtrahend) {
        return minuend - subtrahend;
    }

    public int multiply(final int factor1, final int factor2) {
        return factor1 * factor2;
    }

    public double divide(final int dividend, final int divisor) {
        lastRemainder = dividend % divisor;
        return dividend / divisor;
    }
}
```
MXBean: The More Relaxed Standard MBean

- MXBean can follow Standard MBean naming conventions, but some of them can be relaxed
  - MXBean does not necessarily need to be in interface name
    - Use @MXBean annotation (Java SE 6)
  - MXBean interface and implementation do not need to be in same package
  - Next example demonstrates these
package css2008.dustin.jmx.managed.relaxedmxbean;

import javax.management.MXBean;
@MXBean
public interface CalculatorIf
{
    public int getLastRemainder();

    public int add(final int augend, final int addend);

    public int subtract(final int minuend, final int subtrahend);

    public int multiply(final int factor1, final int factor2);

    public double divide(final int dividend, final int divisor);
}
MXBean Example: Implementation

```java
package css2008.dustin.jmx.managed.relaxedmxbean.impl;

import css2008.dustin.jmx.managed.relaxedmxbean.CalculatorIf;

public class Calculator implements CalculatorIf {
    private int lastRemainder = 0;
    public int getLastRemainder() { return lastRemainder; }
    public int add(final int augend, final int addend) {
        return augend + addend;
    }
    public int subtract(final int minuend, final int subtrahend) {
        return minuend - subtrahend;
    }
    public int multiply(final int factor1, final int factor2) {
        return factor1 * factor2;
    }
    public double divide(final int dividend, final int divisor) {
        lastRemainder = dividend % divisor;
        return dividend / divisor;
    }
}
```
Real Power of the MXBean

- MXBeans provide use OpenTypes that generic clients can understand
  - No need for JMX client to be access custom classes directly in client code
    - Especially useful for JMX clients that cannot be modified or extended
  - No need to include custom classes on JMX client classpath
  - Increases generic JMX client interoperability
About Next MXBean Example

- Will add a Java enum to Calculator MBean classes
  - Standard MBean
  - MXBean

- Will compare what JConsole "sees" in each type of MBean for the exposed enum
  - JConsole is a generic JMX client
Custom Enum for Next Example

```java
public enum CalculatorOperationTypeEnum {
    ADDITION,
    DIVISION,
    MULTIPLICATION,
    SUBTRACTION
}
```
Standard MBean with Enum

public interface CalculatorMBean {
    public int getLastRemainder();
    public CalculatorOperationTypeEnum getLastOperation();
    public int add(final int augend, final int addend);
}

public class Calculator implements CalculatorMBean {
    private int lastRemainder = 0;
    private CalculatorOperationTypeEnum lastOperationType;
    public CalculatorOperationTypeEnum getLastOperation() {
        return lastOperationType;
    }
    public int getLastRemainder() {
        return this.lastRemainder;
    }
}
Note: Enum-based operation is “Unavailable” as attribute in the JConsole MBeans tab.
MXBean with Enum

```java
public interface CalculatorMXBean {
    public int getLastRemainder();
    public CalculatorOperationTypeEnum getLastOperation();
    public int add(final int augend, final int addend);
}

public class Calculator implements CalculatorMXBean {
    private int lastRemainder = 0;
    private CalculatorOperationTypeEnum lastOperationType;
    public CalculatorOperationTypeEnum getLastOperation() {
        return lastOperationType;
    }
    public int getLastRemainder() {
        return this.lastRemainder;
    }
}```
Note: The MXBean mapped the custom enum to an OpenType that JConsole could understand without it being on classpath or in code.
Platform MXBeans

- ClassLoadingMXBean – JVM Class Loading
- CompilationMXBean – JVM Compilation System
- GarbageCollectorMXBean – JVM Garbage Collection
- MemoryManagerMXBean – JVM Memory Manager
- MemoryMXBean – JVM Memory System
- MemoryPoolMXBean – JVM Memory Pool
- OperatingSystemMXBean – OS Hosting JVM
- RuntimeMXBean – JVM Runtime System
- ThreadMXBean – JVM Threading System
- LoggingMXBean – Control logging level (java.util)

Initially available in J2SE 5 – See java.lang.management package
Which MBean to Use?

- Simplicity
  - MXBean if using Java SE 6 or later
  - Standard MBean for pre-Java SE 6

- Generic Client Support without rich description
  - MXBean

- Generic Client Support with rich description
  - Open MBean or MXBean explicit MBeanInfo

- Bridging non-JMX classes with management
  - Model MBeans

- Open MBeans and Model MBeans are both types of Dynamic MBeans
All MBeans Have Descriptors

- Only Model MBeans had Descriptor in J2SE 5
- Java SE 6 introduced Descriptors for all types of MBeans
- Descriptors allow expression of arbitrary data using name-value pairs
- JMX 1.3 began requirement for all MBeans types to support Descriptors
Select New Java SE 6
JMX Classes

- `javax.management.JMX`
  - Convenience class providing static methods such as `newMBeanProxy` and `newMXBeanProxy`

- `javax.management.MXBean`
  - `@MXBean` annotation

- `javax.management.StandardEmitterMBean`
More New Java SE 6 JMX Classes

- `javax.management.remote.JMXAddressable`
- `javax.management.ImmutableDescriptor`
  - Immutable Descriptor
JMX 2.0 (JSRs 255/262)

- **JSR 255 ("JMX 2.0")**
  - Some or all features may be included with Java SE 7 (JSR 255 was slated for SE 6)

- **Possible Enhancements in JMX 2.0**
  - Namespaces / Cascading / Federation
  - JMX Annotations
  - Web Services Connector (JSR 262)
  - Enhanced JMX Querying Capabilities
  - Generic Notifications Filter
  - Notification/Event Service
JMX 2.0: Cascading

- Federation of multiple MBeanServers
- Associate multiple MBean servers in same or different JVMs
- Available today in OpenDMK
- Potential Uses
  - Aggregate multiple JMX servers for single client (simplify client access)
  - Control manageable/monitoring resources for different clients
JMX 2.0 Annotations

- Spring Framework was starting point for JSR-255 annotations discussions
- Allow Standard/MXBeans to be defined with single annotated Java object
  - No need for interface/implementation naming conventions
- Apply more meaningful descriptive information to Standard/MXBeans
- Potentially less verbose code
JMX 2.0 Annotations

Example: Top

@MBean
@Description("Example of using JMX 2.0/JSR-255 JMX annotations to expose " + "an otherwise normal class as a Standard MBean.")
public class Calculator
{
  @ManagedOperation(impact=INFO)
  @Description(
      "Integer Addition: First parameter is the augend and "
      + "second parameter is the addend."
  )
  @DescriptorFields({"p1=augend","p2=addend"})
  public int add(final int augend, final int addend)
  {
    return augend + addend;
  }
}
JMX 2.0 Annotations

Example: Middle

```java
@ManagedOperation(impact=INFO)
@Description(
    "Integer Subtraction: First parameter is minuend "
    + "and second parameter is subtrahend."
)
@DescriptorFields({"p1=minuend","p2=subtrahend"})
public int subtract(final int minuend,
                    final int subtrahend)

{
    return minuend - subtrahend;
}
```
JMX 2.0 Annotations

Example: Bottom

```java
@ManagedOperation(impact=INFO)
@Description(
    "Integer Multiplication: First parameter is one factor "
    + "and second parameter is other factor.")
@DescriptorFields({"p1=factor1","p2=factor2"})
public int multiply(final int factor1, final int factor2)
{ return factor1 * factor2; }

@ManagedOperation(impact=INFO)
@Description(
    "Integer Division: First parameter is the dividend "
    + "and second parameter is divisor.")
@DescriptorFields({"p1=dividend","p2=divisor"})
public double divide(final int dividend, final int divisor)
{ return dividend / divisor; }
```
JMX 2.0 Annotation-Driven MBean in JConsole

![JMX 2.0 Annotation-Driven MBean in JConsole](image.png)

- **Name**: add
- **Description**: Integer Addition: First parameter is the augend and second parameter is the addend.
- **Impact**: INFO
- **ReturnType**: int

**Operation: add**
- **Parameter 1**: p1, 0
- **Parameter 2**: p2, 0

**Augend**:
- p1 as Input

**Addend**:
- p2 as Input
JMX 2 Query Additions

- Easier to query for MBean object names
- Simpler syntax for Query Language
- Syntax based on SQL WHERE clause syntax
- `Query.toString()` provides JMX Query as String
  - Can be reconstituted as JMX Query with `Query.fromString()`
JMX Query Examples

```java
final QueryExp queryExp =
    Query.and(
        Query.eq(
            Query.attr("Priority"),
            Query.value(2) ),
        Query.eq(
            Query.attr("Status"),
            Query.value(StatusEnum.FAILED.toString()) ) );
```

```
final QueryExp queryExp =
    Query.fromString("Priority = 2 and Status = "+ StatusEnum.FAILED.toString()+ ");
```

JMX 1.x Style

JMX 2.0 Style
JMX 2 Notifications Additions

- New Event Service
  - Separates JMX Notification handling from connector
  - JMX Notifications can be sent over alternative connectors

- QueryNotificationFilter
  - Filter JMX Notifications based on JMX Query Language

- NotificationManager interface
Remote JMX Before Web Services Connector

- JSR-160 (“JMX Remote API”)
- Adapters
  - Adapt representation of MBeans to protocol-based view
  - Often heavily tied to underlying protocol
- Connectors
  - Connector Server and Connector Client
  - Client-side interface matches server-side
  - Protocol-specific, but it matters little to client
JMX Adapters and Connectors

- Best-known JMX Adapter
  - OpenDMK HTML Adapter
  - Used extensively in early JMX books

- Best-known JMX Connectors
  - Platform server’s RMI connector (required)
  - JMXMP connector (optional)
    - OpenDMK provides an implementation
  - JSR-262 WS-JMX Connector
```java
final MBeanServer mbs = ManagementFactory.getPlatformMBeanServer();
final JMXServiceURL jmxServiceUrl = new JMXServiceURL(serviceUrl);
final JMXConnectorServer connectorServer =
    JMXConnectorServerFactory.newJMXConnectorServer(jmxServiceUrl, // this differentiates null,
        mbs);
connectorServer.start();
```
General JMX Connector Client

```java
final JMXServiceURL jmxUrl =
    new JMXServiceURL(jmxServiceUrl);

final JMXConnector jmxConnector =
    JMXConnectorFactory.connect(jmxUrl);

final MBeanServerConnection mbsc =
    jmxConnector.getMBeanServerConnection();
```
RMI Connector: JMXServiceURL and More

- JMXServiceURL similar to this:

- Only required connector for JMX implementations

- Supplied with the Java SE JMX

- Requires running RMI registry
  - `rmiregistry` command
JMXMP Connector: JMXServiceURL and More

- JMX Message Protocol
- JMXServiceURL similar to this: `service:jmx:jmxmp://localhost:1098`
  - Described in JMX Specification
  - Optional for JMX implementations
- Not provided by Java SE implementation
- Implementation available in OpenDMK
JMX Web Services Connector

- Benefits of JMX WS Connector:
  - HTTP connectivity is firewall-friendly
  - Non-Java clients can connect to, manage and monitor JMX-instrumented applications

- Web Services Connector Details
  - Supports WS-Management Clients
    - Wiseman, WinRM, Openwsman, etc.
JMX and WS-Management

- WS-JMX Depends on WS-Management
- WS-Management Characteristics
  - DMTF Specification
  - SOAP-based protocol
  - Uses other WS-* specifications
    - WS-Addressing (wsa)
    - WS-Transfer (wsf)
    - WS-Enumeration (wsen)
    - WS-Eventing (wse)
JMX WS Concerns

- Passed May 2008 Public Review Ballot in 6-5 vote
- Several concerns expressed:
  - Proprietary / Immature Underlying Specifications (Oracle)
  - No endpoint WSDL and not WS-I Basic Compliant (IBM)
  - “Yes” voters typically voted “yes” contingent on cited issues being addressed
WS-JMX JMXServiceURL similar to:

-service:jmx:ws://localhost:1097/jmxws

Include ws-jmx-connector JARs on classpath of server and client

-Likely to be unnecessary when standard part of Java SE

Need to use JAX-WS version newer than that currently delivered with Java SE
JMX Web Services Java Client

- Java/Wiseman Client
  - Wiseman handles WS-Management details
  - Client code is exactly the same as for other JMX Connector Clients
  - Must include JSR-262 libraries on client’s classpath
    - Many JARs, including: jmxws.jar, wiseman-core.jar, jaws-rt.jar, jaxws-tools.jar, JAXB jars, JSR 173, JSR 181, JSR 250, many more ...
    - Java SE 6’s JAR wildcard (*) classpath support useful for this
final JMXConnectorServer connectorServer =
    JMXConnectorServerFactory.newJMXConnectorServer(
        new JMXServiceURL(
            "service:jmx:ws://localhost:1097/jmxws"),
        null, mbs);
connectorServer.start();

final JMXConnector jmxConnectorClient =
    JMXConnectorFactory.connect(
        new JMXServiceURL(
            "service:jmx:ws://localhost:1097/jmxws"));
final MBeanServerConnection mbsc =
    jmxConnectorClient.getMBeanServerConnection();

Wiseman / Java-based connector looks like other JMX connectors!
WinRM is WS-Management implementation
WinRM is provided for Windows operating systems
Manage JMX-instrumented Java applications (including JVM) from Windows Operating Systems via WS-JMX
Next example will be based on Windows Vista WinRM client
JMX Web Services Example: Out-of-the-Box WinRM

```
winrm get

-fragment://:Property[@name="LastOperationType"]/*/text()
-r:http://127.0.0.1:1097/jmxws -a:None
```

Using WinRM with JMX Web Services Connector

```
C:\css2008\jmx\MBeanExamples\scripts>winrm get http://jsr262.dev.java.net/DynamicMBeanResource?ObjectName=css2008-calculator-jmxws:type=mxbean -fragment://:Property[@name="LastOperationType"]/*/text()
-r:http://127.0.0.1:1097/jmxws -a:None

XmlFragment = DIVISION

C:\css2008\jmx\MBeanExamples\scripts>
```
JMX Web Services Example: Out-of-the-Box WinRM, 2

WinRM Script Invoking “add” Operation

```
winrm invoke Invoke
-file:C:\css2008\jmx\jmxwsinput\AddOperation.xml
-r:http://127.0.0.1:1097/jmxws -a:None
```

AddOperation.xml Referenced in WinRM Script

```
<ManagedResourceOperation
    xmlns="http://jsr262.dev.java.net/jmxconnector"
    name="add">
    <Input>
        <Param name="augend"><Int>13</Int></Param>
        <Param name="addend"><Int>4</Int></Param>
    </Input>
</ManagedResourceOperation>
```
JMX Web Services Example: Out-of-the-Box WinRM, 3

Command Prompt

C:\css2008\jmx\MBeanExamples\scripts>winrm invoke Invoke http://jsr262.dev.java.net/DynamicMBeanResource?ObjectName=css2008-calculator-jmxws:type=mxbean -file:C:\css2008\jmx\jmxwsinput\AddOperation.xml -r:http://127.0.0.1:1097/jmxws -a:None

ManagedResourceOperationResult
Int = 17

C:\css2008\jmx\MBeanExamples\scripts>
Spring and JMX: Too Easy!

- Spring Framework brings many advantages to JMX development
  - Exposure of non-JMX classes as MBeans
  - Simplifies Connector wiring
  - Simplifies JMX Notifications
  - Provides rich metadata via annotations
    - Inspired currently planned JMX 2.0 annotations
  - If you use Spring, use Spring’s JMX!

Advantages of Model MBeans without all of the effort!
Spring JMX Example: Overview

- Start with non-JMX class (AKA POJO)
  - css2008.dustin.jmx.pojo.Calculator
- Expose POJO as MBean via Spring configuration
  - Using interfaces
  - Using management interface method names
  - Using automatic annotation-based detection
- (Optional) Expose for remote management
Exposing POJO as MBean: Using Interfaces - XML

```xml
<bean id="exposedModelMBean"
class="css2008.dustin.jmx.pojo.Calculator" />

<util:map id="exposedMBeans">
  <entry key="css2008-calculator:type=spring-simple"
         value-ref="exposedModelMBean" />
</util:map>

<bean class="org.springframework.jmx.export.MBeanExporter"
p:beans-ref="exposedMBeans"
p:assembler-ref="assembler" />

<util:list id="manageableInterfaces">
  <value>css2008.dustin.jmx.pojo.CalculatorIf</value>
</util:list>

<bean id="assembler"
class="org.springframework.jmx.export.assembler.InterfaceBasedMBeanInfoAssembler"
p:managedInterfaces-ref="manageableInterfaces" />
```
Exposing POJO as MBean: Using Interfaces - Java

```
final ApplicationContext context =
    new ClassPathXmlApplicationContext(
        "spring-mbean-simple-context.xml");
```

**NOTE:** Not much effort is required to expose any arbitrary Java POJO as a Model MBean. Most of the effort is in XML rather than in Java.
Exposing POJO as MBean: Using Interfaces - JConsole

Model MBean Exposed But No Metadata
Exposing POJO as MBean: Using Metadata – Annotate

```java
@ManagedResource(
    objectName="css2008-calculator:type=spring-metadata",
    description="A simple integer calculator." )
public class Calculator
{
    @ManagedOperation(description="Integer Addition")
    @ManagedOperationParameters({
        @ManagedOperationParameter(
            name="augend",
            description="The first parameter in the addition (augend)."),
        @ManagedOperationParameter(
            name="addend",
            description="The second parameter in the addition (addend)."))
    public int add(final int augend, final int addend)
    {
        return augend + addend;
    }
    ...
```
Exposing POJO as MBean: Using Metadata - XML

```xml
<bean id="exposedModelMBean"
     class="css2008.dustin.jmx.managed.model.spring.Calculator" />

<bean id="exporter"
     class="org.springframework.jmx.export.MBeanExporter"
     p:autodetect="true"
     p:assembler-ref="assembler"
     p:namingStrategy-ref="namingStrategy" />

<bean id="namingStrategy"
     class="org.springframework.jmx.export.naming.MetadataNamingStrategy"
     p:attributeSource-ref="attributeSource" />

<bean id="attributeSource"
     class="org.springframework.jmx.export.annotation.AnnotationJmxAttributeSource" />

<bean id="assembler"
     class="org.springframework.jmx.export.assembler.MetadataMBeanInfoAssembler"
     p:attributeSource-ref="attributeSource" />
```
Exposing POJO as MBean: Using Metadata - Java

```java
final ApplicationContext context =
    new ClassPathXmlApplicationContext(
        "spring-mbean-metadata-context.xml");
```

**NOTE:** As with the other Spring example (interface-based exposure of MBeans), the lines of Java for actually MBean exposure is significantly less than the lines of XML code.

**SECOND NOTE:** Cost of this approach is that Calculator class loses its “pure POJO” status because it is annotated to tell Spring which metadata to expose.
Exposing POJO as MBean: Using Metadata - JConsole

Annotation-based metadata have added descriptive details
**Apache Commons Modeler**

- Use XML to specify Model MBean descriptive metadata

- Is this good or bad? It depends.
  - Advantageous if you prefer XML over Java code
    - Externally configurable metadata
  - Disadvantageous if you prefer Java over XML
    - Must maintain both XML and Java
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mbeans-descriptors PUBLIC
 "-//Apache Software Foundation//DTD Model MBeans Configuration File"
 "http://jakarta.apache.org/commons/dtds/mbeans-descriptors.dtd">
<mbeans-descriptors>
  <mbean id="managedSimpleCalculatorBean"
    name="ManagedSimpleCalculatorBean"
    description="JMX Model MBean with Apache Commons Modeler"
    type="css2008.dustin.jmx.pojo.Calculator">
    <descriptor>
      <field id="nameField" name="name"
         value="ModelMBeanInCommonsModeler" />
      <field id="descriptorTypeField" name="descriptorType"
         value="mbean" />
    </descriptor>
  </mbean>
</mbeans-descriptors>
Commons Modeler Example:
Metadata in XML, 2

```xml
<operation id="addOperation"
  name="add"
  description="Integer Addition"
  impact="INFO"
  returnType="int">
  <parameter id="augendParameter"
    name="augend"
    description="The first parameter in the addition (augend)."
    type="int" />
  <parameter id="addendParameter"
    name="addend"
    description="The second parameter in the addition (addend)."
    type="int" />
</operation>
</mbean>
</mbeans-descriptors>
```
Commons Modeler Example: Used in Java

```java
final css2008.dustin.jmx.pojo.Calculator calculator =
    new css2008.dustin.jmx.pojo.Calculator();
Registry registry = null;
final InputStream modelerXmlInputStream =
    CommonsModelMBeanDemonstrator.class.getResourceAsStream(
        "simple-calculator-modeler.xml");
registry = Registry.getRegistry(null, null);
registry.setMBeanServer(
    ManagementFactory.getPlatformMBeanServer());
registry.loadMetadata(modelerXmlInputStream);
registry.registerComponent(
    calculator,
    objectNameStr,  // mbean registered object name
    "css2008.dustin.jmx.pojo.Calculator");
```

Output in JConsole is same as Model MBeans coded by hand and as Spring Metadata-driven
Easy MBean

- Spring-inspired JMX Annotations without Spring
- Does not require Spring Framework
- Simple to use
- Not as comprehensive as Spring’s JMX annotations support
- Closely associated with jManage
Easy MBean Example: Annotated Java Class, 1

```java
@ManagedResource(
    name="css2008-calculator:type=easymbean",
    description="A simple integer calculator."
)
public class Calculator {

    @ManagedConstructor(
        name="Calculator",
        description="No argument constructor"
    )
    public Calculator() {
    }
}
```
Easy MBean Example:
Annotated Java Class, 2

```java
@ManagedOperation(
    name="add",
    displayName="Integer Addition",
    presentationString="Integer Addition",
    targetClass="Calculator",
    description="Integer Addition")
public int add(final int augend, final int addend) {
    return augend + addend;
}
```
Easy MBean Example: Java MBean Registration

css2008.dustin.jmx.managed.model.easymbean.Calculator mbean =
    new css2008.dustin.jmx.managed.model.easymbean.Calculator();

final ObjectInstance objInstance =
    EasyMBean.getMBean(
        mbean,        // annotated class to be registered MBean
        objectName,   // ObjectName of registered MBean
        mbs);         // MBeanServer on which to register MBean
Easy MBean allows for some metadata to be specified via annotations similar to Spring. However, operations parameters cannot be specified.
State of the Generic JMX Consoles

Current, popular generic JMX consoles:

- **JConsole**
  - Provided with Sun’s JDK since J2SE 5

- **VisualVM**
  - Provided with Sun’s JDK since Java SE 6 Update 7.
  - Available separately as open source download

- **jManage**
  - Open source download
Other Popular JMX Consoles

- **MC4J**
  - One of the first highly visual, generic JMX consoles
  - Seemed neglected for a while

- **OpenDMK HTML Adaptor**
  - Popular in most JMX books
  - Simple web page view into JMX via the HTML Adaptor
  - Originally part of JMX Reference Implementation
  - Now part of OpenDMK project
JConsole: Demonstration

Live Demonstration
VisualVM

- Included with Sun Java SE 6 Update 7
- Consolidates several performance and diagnostic tools
  - jconsole – JMX Console for Java SE 5/6
  - jstat – JVM Statistics Monitoring Tool
  - jstack – Stack trace of Java process
  - jinfo – Configuration of Java process
  - jmap – Memory map of Java process
- Adds graphical features to these tools
VisualVM Demonstration

Live Demonstration
jManage

- **Open Source Application Management**
  - Built on other open source tools
  - Provides web interface and command-line interfaces
    - Web interface is firewall-friendly
    - Command-line interface supports scripting
  - Can be used for building custom consoles
  - Can be used with non-JMX applications
  - JFreeChart-based graphs
## jManage Demonstration

The jManage Management Console - Mozilla Firefox shows the following applications and components:

**JMXImplementation**
- `type=MBeanServerDelegate`

**com.sun.management**
- `type=HotSpotDiagnostic`

**dustin**
- `type=alternate.name=Another`
- `type=complex.name=One`
- `type=complex.name=Two`
- `type=simple.name=One`
- `type=simple.name=Two`

**java.lang**
- `type=ClassLoading`
- `type=Compilation`
- `type=GarbageCollector.name=Copy`
- `type=GarbageCollector.name=MarksweepCompact`
- `type=Memory`
- `type=MemoryManager.name=CodeCacheManager`
- `type=MemoryPool.name=Code Cache`
- `type=MemoryPool.name= Eden Space`
- `type=MemoryPool.name= Perm Gen`
- `type=MemoryPool.name= Perm Gen (shared-ro)`
**jManage CLI Demonstration**

```
C:\jmanage-2.0-RC1\bin>jmanage -username admin -password 123456
Oct 18, 2008 10:13:45 PM org.jmanage.core.util.CoreUtils <clinit>
INFO: JManage.root=..
jmanage>help
jmanage [-username <username>] [-password <password>] [-verbose=[<level>]] [command] [command args]

Commands:
  apps     Lists all configured applications in jManage
  cmbeans  Lists configured mbeans for the given application
  execute  Executes given operation on mbean.
  exit     Exits from jManage command prompt mode
  get      Gets attribute values for given mbean.
  help     Prints JManage command line help
  info     Display information about the mbean.
  mbeans   Queries mbeans for the given application
  print    Prints tab delimited value(s) for given mbean attribute(s).
  set      Sets attribute value.
  setattrs Sets one or more attribute values.

Type "help <command>" for detailed command help.
[Important! : Any space within a command argument should be replaced with "\" character.]

jmanage>apps
jManage [local] null
Oracle Database Connection [connector] null
rmi
```

**NEW**

Dustin Marx — Java Management Extensions (JMX) Circa 2008
jManage CLI Demonstration

```
jmanage> mbeans JMX~Querying~Example

javax.management:type=MBEanServerDelegate
com.sun.management:type=HotSpotDiagnostic
dustin:type=alternate,name=Another
dustin:type=complex,name=One
dustin:type=complex,name=Two
dustin:type=simple,name=One
dustin:type=simple,name=Two
java.lang:type=ClassLoader
java.lang:type=Compilation
java.lang:type=GarbageCollector,name=Copy
java.lang:type=GarbageCollector,name=MarkSweepCompact
java.lang:type=Memory
java.lang:type=MemoryManager,name=CodeCacheManager
java.lang:type=MemoryPool,name=Code Cache
java.lang:type=MemoryPool,name=Eden Space
java.lang:type=MemoryPool,name=Perm Gen
java.lang:type=MemoryPool,name=Perm Gen [shared-ro]
java.lang:type=MemoryPool,name=Perm Gen [shared-rw]
java.lang:type=MemoryPool,name=Survivor Space
java.lang:type=MemoryPool,name=Tenured Gen
java.lang:type=OperatingSystem
java.lang:type=Runtime
java.lang:type=Threading
java.util.logging:type=Logging

jmanage>
```
GlassBox Inspector

- Uses JMX with Aspect-Oriented Programming (AspectJ)
- Useful for identifying performance issues and diagnosing cause
- Minimal impact on executing code performance
- Easy installation
Project OpenDMK

- Non-standard functionality from original JMX Reference Implementation
  - Standard functionality left in current RI
- Legacy Connectors and HTML Adaptor
- Federated MBean Server / Cascading
- REST-style Web Services Connector (in-progress)
- SNMP support
JMX Lessons Learned: Best Practices

- Separate management and business logic
- Consider interoperability
  - MXBeans, OpenMBeans, JMX Web Services Connector
- Naming conventions
  - Object Names, Notifications
- Understand limitations of JMX Notifications
- See also “Advanced Sample” included with Sun’s Java SE 6 JDK
JMX Books Are Aging

- JMX books are relatively old (2002-2004 mostly)
  - Fundamentals are mostly the same
  - Light or non-existence coverage of modern JMX features:
    - MXBeans
    - Remote JMX
    - Modern JMX Consoles
    - JMX 2 features / Web Services for JMX Connector
- Topics covered in this presentation!
So Where Do I Get Current JMX Information?

- **Online JMX Resources**
  - JMX Technology Home Page
  - List on Online JMX Resources
    - [http://marxsoftware.blogspot.com/2008/03/online-jmx-resources.html](http://marxsoftware.blogspot.com/2008/03/online-jmx-resources.html)

- **Sun JMX Forum**
Blogs May Be Best Source for Latest JMX News

- Relevant and Useful JMX Blogs
  - Eamonn McManus
    - http://weblogs.java.net/blog/emcmanus/
  - Daniel Fuchs
  - Luis Miguel-Alventosa
  - Jean-Francois Denise
  - My Blog (JMX entries)
    - http://marxsoftware.blogspot.com/search/label/JMX
The State of JMX in 2008

- JMX has come a long way since JSR-3
- JMX in 2008 is
  - **Simpler**
    - Included with Sun’s J2SE 5 / Java SE 6
    - MXBeans, Spring, Easy MBean, Commons Modeler
  - **More Open**
    - JMX Web Services Connector, MXBeans
  - **More Useful**
    - JConsole, VisualVM, ...
Background Slides

- Additional References
  - Introductory JMX Resources
  - Remote JMX (Except Web Services Connector)
  - JMX 2.0
  - JMX Web Services Connector
  - JConsole / Visual VM

- Questions and Comments
  - Asked in the three presentations of these slides
Additional Resources:
Introductory JMX Resources

- Java SE Tutorial – JMX Thread
- What is JMX?
  - [http://blogs.sun.com/jmxetc/entry/what_is_jmx](http://blogs.sun.com/jmxetc/entry/what_is_jmx)
- Online JMX Resources
  - [http://marxsoftware.blogspot.com/2008/03/online-jmx-resources.html](http://marxsoftware.blogspot.com/2008/03/online-jmx-resources.html)
- JMX Specifications: An Overview
  - [http://marxsoftware.blogspot.com/2008/03/jmx-specifications-overview.html](http://marxsoftware.blogspot.com/2008/03/jmx-specifications-overview.html)
Additional Resources: Remote JMX

- Remote JMX: Connectors and Adapters

- Remote JMX Exceptions
  - [http://marxsoftware.blogspot.com/2008/06/remote-jmx-exceptions.html](http://marxsoftware.blogspot.com/2008/06/remote-jmx-exceptions.html)

- Simple Remote JMX with GlassFish
  - [http://marxsoftware.blogspot.com/2008/05/simple-remote-jmx-with-glassfish.html](http://marxsoftware.blogspot.com/2008/05/simple-remote-jmx-with-glassfish.html)

- Troubleshooting JConsole Connection Problems
Additional Resources:
JMX 2.0 (Java SE 7)

- Playing with the JMX 2.0 API
  - [http://weblogs.java.net/blog/emcmanus/archive/2008/08/playing_with_th.html](http://weblogs.java.net/blog/emcmanus/archive/2008/08/playing_with_th.html)

- Dropping Proposed Features from the New JMX API
  - [http://weblogs.java.net/blog/emcmanus/archive/2008/09/dropping_propos.html](http://weblogs.java.net/blog/emcmanus/archive/2008/09/dropping_propos.html)

- Playing with JMX 2.0 Annotations

- Querying in JMX 2.0
Additional Resources:
JMX Web Services Connector

- ws-jmx-connector
  - https://ws-jmx-connector.dev.java.net/

- JMX Interoperation with Non-Java Technologies

- Three Looks at JMX Web Services Connector
  - http://marxsoftware.blogspot.com/2008/05/first-look-at-jmx-web-services.html
Additional Resources: JConsole / VisualVM

- Chuk-Munn Lee of Sun Microsystems Troubleshoots Java SE 6 Deployment

- Using JConsole to Monitor and Manage Java Applications

- Introduction to VisualVM
  - [https://visualvm.dev.java.net/description.html](https://visualvm.dev.java.net/description.html)
Questions / Comments

Q: Do the properties need to be capitalized in JMX 2.0 Query Expressions because of a JavaBean convention?

A: Don’t know but that seems like a reasonable explanation
Questions / Comments

Q: Can I use JConsole in a 1.5 JVM to monitor/manage an instrumented application in Java SE 6 that uses Java 6-specific features?

A: Generally, yes. The important point this question brings up is that JConsole 1.5 can monitor Java SE 6-based JMX applications, but it cannot take advantage of any of the new features.
Q: If a class using JMX 2/Java SE 7 marks a particular class with the @MBean annotation, are all methods exposed or just the methods annotated with @ManagedOperation?

A: Fortunately, only the methods annotated with @ManagedOperation are exposed when the class is annotated with @MBean.