Spring and JMX

Manageability for Spring-based Applications

Scott W. Ruch
Inventa Technologies
www.inventa.com
Goals

- Understand why manageability is important
- Review the basics of JMX
- Learn how to leverage Spring’s simple and transparent JMX integration
Topics

- Introduction
- Management
- JMX Overview
- Hosting Spring in JBoss
- Spring JMX Support
- Q&A
Introduction

Scott W. Ruch
sruch@inventa.com

- Software Architect, Inventa Technologies
- Software Architect, HP-Bluestone Middleware Division
  - I18n, Manageability
  - JSR 77, 160 Expert Group Member
- Application Server Technology (1999)
- Java Technology (1997)
- Telecommunications, Distributed Object Systems
- UNIX Kernel, Device Drivers, Communications Protocols
- I/O Subsystems Hardware Design / Qualification
Who We Are

- Performance Management
- Monitoring
- Capacity Planning

- Partner with ESM Vendors (BMC, HP, CA,...)
  - Help customers leverage software investments
  - Integrate
  - Collect Management Data
  - Analysis / Planning / Reporting

- ESM Integrator™ Platform
  - Service / Product
ESM Integrator™ Datahub

Sybase

PATROL

CHE

DTE

Datahub

Data Objects

CHE Collectors

Local Collector

Data Objects

Data Message

Data Sources

Rule Engine

Filter Engine

SNMP Trap

Alert Loader

Master DB

Data Loaders

Datastore A

Datastore B

Datastore C

Data Statistics

SNMP Traps
Management
Management: What is it?

- Configuring
  - Initialization (DI)
  - Runtime / Operational

- Controlling
  - Life-cycle: start, stop, suspend, resume
  - Availability (load-balancing, capacity)

- Monitoring
  - Performance
  - Health
  - State
Why is Manageability Important?

- `logger.setLevel(Level.DEBUG)`?

- Analogy: Internationalization
  - Retrofitting can be difficult
  - Cost prohibitive

- Static configurations are limiting

- Integration
  - Expose counters / operations / state
  - Event notification
  - Remote management
JMX Overview
JMX: A Brief History

- **JSR 3: Java Management eXtensions**
  - Initiated: 1998
  - Precursors: JMAPI, JDMK
  - 1.0 RI released: 12/2000
  - 1.2 → Java 1.5

- **JSR 160: JMX Remote API**
  - Initiated: 2001
  - RMI + Generic Protocol (JMXMP)
  - 1.0 Release: 10/2003
  - 1.0 → Java 1.5
JMX: A Brief History

- JSR 77: J2EE Management
  - Initiated: 2000
  - Standard J2EE management model for
    - Server resource management
    - Application lifecycle control
    - Performance monitoring
  - Defines:
    - Managed Object naming conventions
    - “Standard” J2EE Managed Objects
  - Access via:
    - Management EJB (MEJB) (required)
    - SNMP Agent (optional)
    - CIM / WBEM (optional)
  - 1.0 → J2EE 1.4
What is JMX?

- Architecture, APIs, Design Patterns

- Components:
  - Managed (Instrumented) Resources
  - Agents (and their services)
  - Connectors / Adapters

- Goals:
  - Instrument Java Code
  - Create Smart Java Agents
  - Implement Management Middleware
  - Integrate with Industry Standard Management Platforms
JMX Summarized

- Resources are instrumented with Management Beans (MBeans)
- MBeans expose the interface that is used to manage the resource:
  - Attributes (getter/setter)
  - Operations
  - Notifications (events)
- MBeans are registered in an agent (`javax.management.MBeanServer`)
- Registered MBeans are visible to local or remote management applications through the agent.
JMX Architecture

- Distributed Services Level
  - Management Application
  - Connector
    - RMI / JMXMP
  - JSR 160

- Agent Level
  - MBean Server
    - Timer
    - Relation
    - Class Loading
    - Monitor
    - Agent Service MBeans

- Instrumentation Level
  - MBean
  - MBean
  - MBean
  - JMX-Managed Resources
  - JVM

- Web Browser
  - HTTP/S
  - JSR 160
  - Connector
  - Protocol Adapter
JMX Architecture

Distributed Services Level
- Management Application
- Connector
  - RMI / JMXMP
- HTTP/S
- JSR 160

Agent Level
- MBean Server
  - Timer
  - Relation
  - Class Loading
  - Monitor
  - Agent Service MBeans

Instrumentation Level
- MBean
- MBean
- MBean
  - JMX-Managed Resources

External Adapters / Tools
- SNMP Manager
- No JSR
- CIM/WBEM
  - JSR 48
  - JSR 146 (withdrawn)
- TMN
  - JSR 71 (withdrawn)
JMX Architecture

Distributed Services Level

Management Application

Connector

RMI / JMXMP

JSR 160

Agent Level

Connector

Protocol Adapter

Web Browser

HTTP/S

Instrumentation Level

MBean Server

Timer
Relation
Class Loading
Monitor

Agent Service MBeans

MBean

MBean

MBean

JMX-Managed Resources

JVM

Distributed Services Level

Web Browser

HTTP/S

Instrumentation Level

Agent Level

Management Application

Connector

RMI / JMXMP

JSR 160

MBean

MBean

MBean

JMX-Managed Resources

JVM
JMX Instrumentation Level

- **Standard MBean**
  - Concrete Java class providing:
    - Implementation of its corresponding static MBean interface
      e.g.: `class Manageable implements ManageableMBean`
    - Optionally, an implementation of the `NotificationBroadcaster` interface

- **Dynamic MBean**
  - Concrete Java class providing:
    - Implementation of the `DynamicMBean` interface
    - Optionally, an implementation of the `NotificationBroadcaster` interface

- Only difference is development model!
Standard MBean

```
«interface»
ManageableMBean
+start()
+stop()
+suspend()
+resume()
+getUptime() : long
+getItemsProcessed() : long
```

```
«interface»
NotificationBroadcaster
```

```
Manageable
+start()
+stop()
+suspend()
+resume()
+getUptime() : long
+getItemsProcessed() : long
```
Dynamic MBean

```java
javax.management.DynamicMBean
+getMBeanInfo() : MBeanInfo
+getAttribute(in attribute : String) : Object
+getAttributes(in attributes : String[]) : AttributeList
+setAttribute(in attribute : Attribute)
+setAttributes(in attributes : AttributeList) : AttributeList
+invoke(in actionName : String, in params : Object[], in signature : String[]) : Object
```

```java
javax.management.Manageable
+getMBeanInfo() : MBeanInfo
+getAttribute(in attribute : String) : Object
+getAttributes(in attributes : String[]) : AttributeList
+setAttribute(in attribute : Attribute)
+setAttributes(in attributes : AttributeList) : AttributeList
+invoke(in actionName : String, in params : Object[], in signature : String[]) : Object
+start()
+stop()
+suspend()
+resume()
+getUptime() : long
+getItemsProcessed() : long
```

«interface» NotificationBroadcaster
Other Dynamic MBeans

- **Open MBeans**
  - “Open” to the widest range of management applications
  - Use a small set of universal Java types.
  - Fully advertise / describe their functionality
  - Useful when management application does not have access to the agent’s Java classes

- **Model MBeans**
  - Truly dynamic
  - Allows “on-the-fly” creation of MBeans
  - Instantiate `RequiredModelMBean`, Configure, Register
JMX Architecture

Distributed Services Level

Management Application

Connector

RMI / JMXMP

Agent Level

Connector

Protocol Adapter

HTTP/S

Instrumentation Level

MBean Server

Timer
Relation
Class Loading
Monitor
Agent Service MBeans

MBean
MBean
MBean

JMX-Managed Resources

JVM

Web Browser
JMX Agent Level

- Foundation Classes
- MBean Server
- Agent Services
Foundation Classes

- API argument types, return values
  - ObjectName
  - ObjectInstance
  - Attribute, AttributeList

- Metadata classes:
  - MBeanInfo
  - MBeanFeatureInfo
  - MBeanAttributeInfo
  - MBeanOperationInfo
  - MBeanConstructorInfo
  - MBeanParameterInfo
  - MBeanNotificationInfo
Object Names

- `javax.management.ObjectName`
- Uniquely identifies MBean within MBeanServer
- Consists of:
  - Domain (optional)
    - If omitted, MBeanServer supplies default
  - Unordered set of one or more property pairs
  - Any characters except ",=*?:
  - Case-sensitive!
  - Syntax: `[domainName]:property=value[,property=value]`*
- Example: `myDomain:type=Service,Service=scheduler`
- Conventions
  - Domain: `com.inventa.services`
  - Include “type” property to aid searches
- Canonical form
  - Properties sorted lexicographically for comparison
Object Names – Pattern Matching

- Domain
  - File globbing conventions (*, ?)

- Properties
  - No wildcard matching on name / value
  - Wildcard for unspecified properties

- Examples
  - All MBeans in all domains: "*:*
  - All MBeans in default domain: "*
  - All MBeans of type “Service”: "*:type=Service,*"
MBean Server

- Registry for MBeans in the agent, provides all services for manipulating MBeans

- `interface javax.management.MBeanServer`

- Implementation(s) obtained via `MBeanServerFactory.findMBeanServer()`

- Object Name: `JMIImplementation:type=MBeanServerDelegate`
MBean Registration

- Instantiate and register
  - `createMBean(String className, ObjectName name)`
  - `createMBean(String className, ObjectName name, Object params[], String signature[])`

- Register pre-existing
  - `registerMBean(Object object, ObjectName name)`

- Unregister
  - `unregisterMBean(ObjectName name)`
MBean Queries

/* Find all the JBoss JMS Queue MBeans */

MBeanServer server = ...

ObjectName name = new
    ObjectName("jboss.mq.destination:service=Queue,*");

Set queueNames = server.queryNames(name, null);
Set queues = server.queryMBeans(name, null);
MBean Operations

- **Metadata**
  - `getMBeanInfo(ObjectName name)`

- **Attributes**
  - `getAttribute(ObjectName name, String attribute)`
  - `getAttributes(ObjectName name, String[] attributes)`
  - `setAttribute(ObjectName name, Attribute attribute)`
  - `setAttributes(ObjectName name, AttributeList attributes)`

- **Operations**
  - `invoke(ObjectName name, String operationName, Object params[], String signature[])`
Method Calls, JMX 1.1

```java
public interface CachingManagerMBean {
    String[] getManagerNames();
    void clearCache(String managerName);
}

MBeanServer server = ... 

ObjectName name = new 
    ObjectName("inventa:type=Service,service=CachingManager");

String[] managers = (String[])server.getAttribute(name, "ManagerNames");

for (String manger : Managers) {
    Object result = server.invoke(
        name, "clearCache",
        new Object[]{manager},
        new String[]{String.class.getName()});
}
```
Method Calls, JMX 1.2

```java
MBeanServer server = ...;

ObjectName name = new ObjectName("inventa:type=Service,service=CachingManager");

CachingManagerMBean cachingManager = (CachingManagerMBean) MBeanServerInvocationHandler.newProxyInstance(
    server,
    name,
    CachingManagerMBean.class,
    false);

String[] managers = cachingManager.getManagerNames();

for (String manager : managers) {
    cachingManager.clearCache(manager);
}
```
Notifications

- MBeans can optionally implement `NotificationBroadcaster`
  - Allows listeners to register with MBean
  - MBeans should implement `getNotificationInfo()`

- In practice: MBeans extend `NotificationBroadcasterSupport`
  - Raise notifications with `sendNotification()`
  - Base class delivers to all registered listeners

- Listeners register via MBeanServer
  - `addNotificationListener(ObjectName name, NotificationListener listener, NotificationFilter filter, Object handback)`
class NotifyingService
    extends NotificationBroadcasterSupport
    implements NotifyingMBean {

    private void raiseAlert() {

        Notification n = new Notification(ALERT_EVENT_TYPE, this,
            getNextNotificationSequenceNumber());

        /* Add notification-specific information */
        Map context = new HashMap();
        context.put(...);
        n.setUserData(context);

        sendNotification(n);
    }
}
Agent Services

- **Timer**
  - Scheduling facility for single or periodic notifications

- **Monitoring**
  - Allows observation of other MBean’s attributes
  - Receive notification when attribute satisfies certain conditions

- **Class Loading**
  - Provides remote class-loading when registering MBeans external to the MBeanServer.

- **Relationship**
  - Allows definition of multiplicity relationships between registered MBeans. (1..1, 1..n, etc)
  - Service maintains consistency, blocks creation of invalid relationships.
  - Relationships are separate from MBean registration – they can be added or removed but the MBeans are unaffected.
JMX Architecture

Distributed Services Level

Management Application

Connector

RMI / JMXMP

JSR 160

Web Browser

HTTP/S

Agent Level

Connector

Protocol Adapter

Instrumentation Level

MBean Server

Timer
Relation
Class Loading
Monitor

Agent Service MBeans

MBean
MBean
MBean

JMX-Managed Resources

JVM
Distributed Service Level

- Connectors
  - JMX (JSR 3) defined the concept
  - JSR 160 defines Connector APIs
    - Default protocols are RMI, JMXMP
    - Other implementations: SOAP, Burlap, Hessian, ...
    - Good news: API independent of serialization protocol / transport

- Client interface: JMXConnector
  - Factory: JMXConnectorFactory

- Server interface: JMXConnectorServer
  - Factory: JMXConnectorServerFactory

- JMXServiceURL identifies server
  - service:jmx:<protocol>://[[<host>]:<port>]/<path>]

9/26/2006 SpringForward 2006 Spring and JMX
Using Connectors

- Connector server is created / started either:
  - Standalone
  - By registering it as an MBean within the target MBeanServer

- Creating connector client:

```java
JMXServiceURL address = new JMXServiceURL("service:jmx:rmi:///remoteHost");
connector = JMXConnectorFactory.connect(address, null);

MBeanServerConnection server =
  connector.getMBeanServerConnection();
```

- `MBeanServer` implements `MBeanServerConnection`
Hosting Spring in JBoss
JBoss 4

- JMX Microkernel, modular architecture
JBoss JMX Console / MBean Services

Demonstration
Spring JMX Support
Spring’s Core JMX Features

- Automatic registration of *any* Spring bean as a JMX MBean
- Flexible control of the management interface
- Flexible control of Object Names
- Declarative exposure of MBeans over remote, JSR-160 connectors
- Proxying of both local and remote MBeans

- New features in Spring 2.0
  - Control of MBean registration behavior
  - Notification support
MBean Registration

- Core class **MBeanExporter**: Registers Spring beans as JMX MBeans

- Explicit registration of any Spring bean:
  - Valid JMX MBeans
  - POJOs

- Automatic detection / registration of valid JMX MBeans
Explicit Registration

```xml
<bean id="exporter"
    class="org.springframework.jmx.export.MBeanExporter"
    lazy-init="false">
  <property name="beans">
    <map>
      <entry
          key="inventa.service:type=Service,service=Manageable"
          value-ref="Manageable"/>
      <entry key="inventa.service:type=Person"
          value-ref="John"/>
    </map>
  </property>
</bean>
```
Automatic Registration

- **MBeanExporter** detects all valid JMX MBeans

```xml
<bean id="exporter"
     class="org.springframework.jmx.export.MBeanExporter"
     lazy-init="false">

   <!-- Auto register any standard MBeans -->
   <property name="autodetect" value="true"/>

</bean>
```
Spring 2.0 Feature – Registration Behavior

- Use `MBeanExporter` “registrationBehaviorName” property

  ```xml
  <property name="registrationBehaviorName" value="REGISTRATION_FAIL_ON_EXISTING"/>
  ```

- `REGISTRATION_FAIL_ON_EXISTING`
- `REGISTRATION_IGNORE_EXISTING`
- `REGISTRATION_REPLACE_EXISTING`
Management Interface Control

- Powerful and comprehensive facilities
- Many ways to control the exposed interface

- `MBeanExporter` delegates to implementations of `MBeanInfoAssembler` for creating management interfaces at runtime
  - Reflection-based (default, exposes everything)
  - Source Level Annotation–based
    - Commons Attributes
    - JDK 5.0 Annotations
  - Java Interface-based
  - Method name-based
Method Name-Based Interface Assembly

<bean id="exporter"
     class="org.springframework.jmx.export.MBeanExporter"
     lazy-init="false">

    <property name="beans">
        <map>
            <entry key="inventa.service:type=Person"
                   value-ref="John"/>
        </map>
    </property>

    <property name="assembler">
        <bean
            class="org.springframework.jmx.export.assembler.MethodNameBasedMBeanInfoAssembler">
            <property name="managedMethods">
                <value>setName,getName,setAge,getAge</value>
            </property>
        </bean>
    </property>

</bean>
Spring MBean Interface Control Demonstration
Object Name Control

- Many ways to control ObjectNames as well...

- `MBeanExporter` delegates to implementations of `ObjectNamingStrategy` for creating ObjectNames
  - Based on key from “beans” property map (default)
  - Based on external Properties file
  - Based on JVM’s notion of object identity
  - Based on source-level annotation
JSR 160 Connectors / Proxies

- Spring JMX supports creating both server and client Connector beans.
- A client Connector bean could point to any remote MBeanServer
- Spring can proxy remote objects:

```xml
<bean id="clientConnector"
     class="org.springframework.jmx.support.MBeanServerConnectionFactoryBean">
  <property name="serviceUrl" value="service:jmx:rmi://remotehost:9875"/>
</bean>

<bean id="proxy"
     class="org.springframework.jmx.access.MBeanProxyFactoryBean">
  <property name="objectName" value="bean:name=testBean"/>
  <property name="proxyInterface" value="org.springframework.jmx.IJmxTestBean"/>
  <property name="server" ref="clientConnector"/>
</bean>
```
Thanks!

Questions?
References / Resources

- Spring Framework 2.0.x Reference: http://static.springframework.org/spring/docs/2.0.x/reference/index.html