Java Management Extensions (JMX)

Basics

Topics Covered

- **Management**
  - Introduction
  - Tasks
  - Operational Models
  - Instrumentation Models
- **Monitoring & Management Features in Java**
- **JMX Introduction**
- **JMX Architecture**
  - Instrumentation Level
  - Agent Level
  - Distributed Services Level
- **Why JMX**
- **Epilogue**

Management of a System

- **Management of System** refers to the activities, methods, procedures, and tools that pertain to the Operation, Administration, Maintenance, and Provisioning of Managed Systems

- Management architectures are becoming increasingly integrated within distributed managed systems.

- The goal of management is to ensure that the managed systems operate with the efficiency and effectiveness predefined in the quality of service parameters.

- They can severely impair the performance, if the management strategy is not well defined and studied.

- Management of a System helps us to figure out what is happening.

Management Tasks

- **Monitoring**: the ability to capture runtime and historical events from a particular component, for reporting and notification purpose. It is a continuous action over the execution time of the Managed System.

- **Tracking**: the ability to observe aspects of a single unit of work or thread of execution across multiple components. It is executed less frequently than the Monitoring.

- **Control**: the ability to alter the runtime behavior of a managed component (e.g., changing the logging level of an application). This task is executed on a precise period from the execution time of the managed system.

Management Operation Models

Polling:

- The collector is invoked on a regular frequency, and it retrieves and traces the current value of a metric or set of metrics from the Managed System.
- For example, a collector might be invoked every minute to read a host's CPU utilization or read the total number of committed transactions from a transaction manager through a JMX interface.
- The premise of a polling pattern is a periodic sampling of a target metric.

Listening:

- This is a sort of Observer pattern.
- The collector registers itself as a listener of events with the target Managed System and receives a callback whenever the event of interest occurs.

Interception:

- In this pattern, the collector inserts itself as an interceptor between a target and its caller or callers.
- On each instance of activity that passes through the interceptor, it makes a measurement and traces it.

Management Instrumentation Models

Instrumentation allows management of Resources. Most Commonly used Management Instrumentation Models:

- – Internal instrumentation and
- – External instrumentation.

- In the **Internal Instrumentation**, the management object is part of the managed resource and management tasks are executed directly on it.
- **External instrumentation** is defined and executed outside the managed resource.
- Internal instrumentation might affect more significantly the performance of the managed resource rather than the external one.

Monitoring & Management Features in Java

Starting with J2SE 5.0, the Java platform includes the following monitoring and management features:

- JVM Instrumentation
- Monitoring and Management APIs
- Tools
- Integrated JMX

Monitoring & Management Features in Java

JVM Instrumentation

The Java virtual machine (JVM) is instrumented for monitoring and management, providing built-in ("out-of-the-box") management capabilities for both remote and local access.

Monitoring and Management APIs

The *java.lang.management* package provides the interface for monitoring and managing the JVM. The API provides access to information such as:

- Number of classes loaded and threads running
- Virtual machine uptime, system properties, and JVM input arguments
- Thread state, stack trace of live threads
- Memory consumption
- Garbage collection statistics
- Operating system information, etc

Monitoring & Management Features in Java

Tools

A graphical JMX monitoring tool, jconsole, enables you to monitor the performance of a JVM and instrumented applications, providing information to help you optimize performance.

Integrated JMX

The JMX API allows you to instrument applications for monitoring and management. The RMI connector allows this instrumentation to be remotely accessible.

JMX Introduction

- Java Management Extensions (JMX) specification defines the Management Architecture, which enables Management and Monitoring of applications and services.
- JMX Architecture provides interfaces for defining runtime Monitoring, Management and Configuration support for applications.
- JMX is a standardized means for providing a remotely accessible management interface and is an easy way to add a flexible and powerful management interface to an application.
- JMX standardizes interfaces for monitoring resources that enable anyone to use an arbitrary technology to build monitoring applications. These applications can easily access the monitored resources communicating through a JMX agent.
- JMX delivers the three basic monitoring modes: sampling (the pull mode), tracing (the push mode) and periodic notification on events. The user is able to choose the most suitable mode of delivery of monitoring data that depends on the application.
- A resource is an entity in the system that needs to be monitoring and/or controlled by a management application; resources that can be monitored and controlled are called manageable. JMX architecture enables Java applications to become manageable.

JMX Architecture

The JMX architecture is based on a manager and agent model.

The JMX Architecture has three layers:

- Instrumentation
- JMX agent
- Distributed Services

The key components of each architectural level are listed below:

Instrumentation level

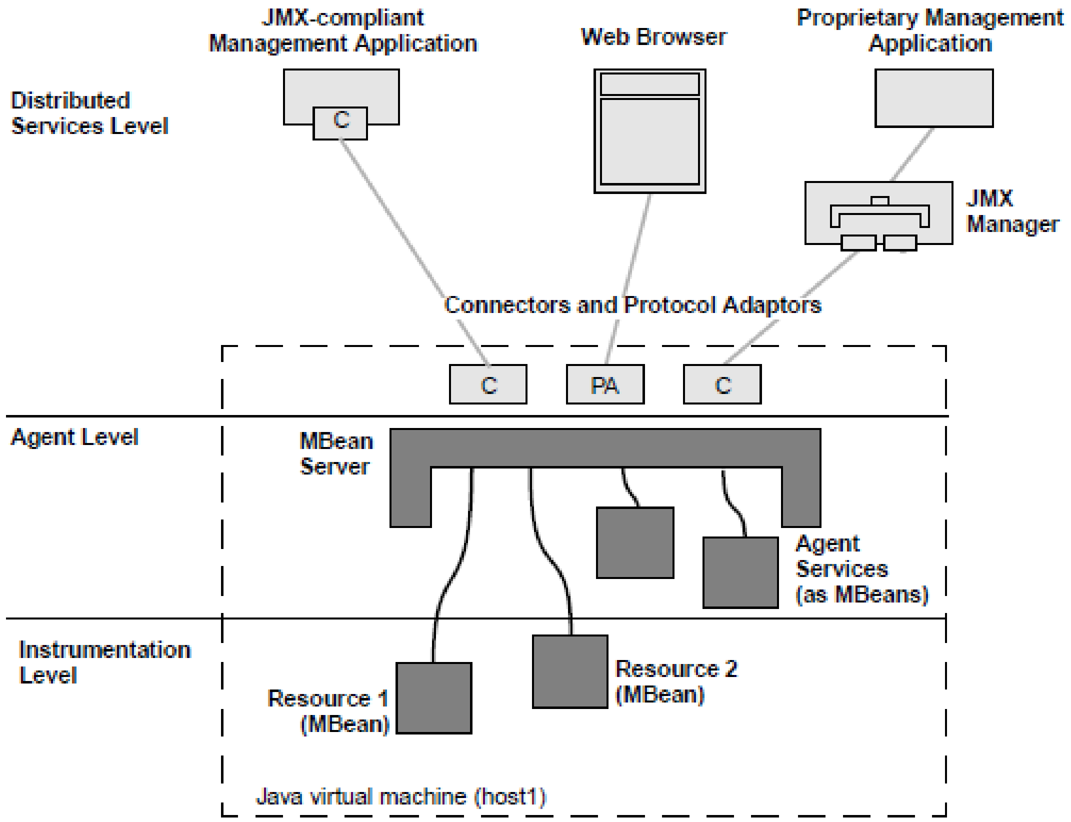
- MBeans (standard, dynamic, open, and model MBeans)
- Notification model
- MBean metadata classes

Agent level

- MBean server
- Agent services

Distributed Services Level

- Connectors and Protocol Adapters
- Management Applications


JMX-compliant Management Application
Web Browser
Proprietary Management Application
Distributed Services Level
C
JMX Manager
Connectors and Protocol Adaptors
C
PA
C
Agent Level
MBean Server
Agent Services (as MBeans)
Instrumentation Level
Resource 1 (MBean)
Resource 2 (MBean)
Java virtual machine (host1)

Instrumentation Level

JMX Architecture → Instrumentation Level

- The Instrumentation level prepares resources to be manageable. This level defines the requirements for implementing JMX manageable resources.

- The management interface of a resource is the set of all necessary information and controls that a management application needs to operate on the resource.

- The instrumentation of a resource allows it to be manageable through the agent level.

- The manageable resource exposes a Java object or wrapper that describes its manageable features, which makes the resource instrumented so that it can be managed by JMX-compliant applications.

- This level provides specification for implementing JMX manageable resources.

Managed Beans (MBeans)

JMX Architecture → Instrumentation Level → MBeans

- The Java objects which implement resources and their instrumentation are called Managed Beans, or MBeans for short.
- MBeans must follow the design patterns and interfaces defined in the JMX specification. Doing so ensures that all MBeans provide managed resource instrumentation in a standardized way.
- MBeans do not require knowledge of the JMX agent with which they operate.
- The process of inspecting the MBean interface and applying the design patterns is called *Introspection*.
- The JMX agent uses introspection to look at the methods and superclasses of a class, determine if it represents an MBean that follows the design patterns, and recognize the names of both attributes and operations.

Managed Beans (MBeans)

JMX Architecture → Instrumentation Level → MBeans

- Objects which are implemented as an MBeans and registered with the agent can be managed from outside the agent's Java virtual machine. Such objects include:
  - The resources your application wishes to manage
  - Value-added services provided to help manage resources
  - Components of the JMX infrastructure that can be managed

- The management interface of a standard MBean is composed of:
  - Its constructors: only the public constructors of the MBean class are exposed
  - Its attributes: the properties which are exposed through getter and setter methods
  - Its operations: the remaining methods exposed in the MBean interface
  - Its notifications: the notification objects and types that the MBean is likely to emit

- Different types of resource can be managed using JMX technology, for example an application, an implementation of a service, a device or a user.

- MBeans can also be used as wrappers for legacy code without a management interface or as proxies for code with a legacy management interface.

- Developers of applications and devices are free to choose the granularity of objects that are instrumented as MBeans.
- An MBean might represent the smallest object in an application, or it could represent the entire application.

Managed Beans (MBeans)

JMX Architecture → Instrumentation Level → MBeans

- JMX defines the following types of MBeans:
  - **Standard MBeans** are the simplest to design and implement, their management interface is described by their method names.
  - **Dynamic MBeans** must implement a specific interface, but they expose their management interface at run-time for greatest flexibility.
  - **Open MBeans** are dynamic MBeans which rely on basic data types for universal manageability and which are self-describing for user-friendliness.
  - **Model MBeans** are also dynamic MBeans that are fully configurable and selfdescribed at run-time; they provide a generic MBean class with default behavior for dynamic instrumentation of resources.
  - **MXBeans** is an MBean that references only a pre-defined set of data types.

- Each of these MBeans corresponds to a different instrumentation need.
- Standard & MXBeans are Static MBeans
- Dynamic, Model & Open are Dynamic MBeans

Notification Model

JMX Architecture → Instrumentation Level → Notification Model

- The JMX specification defines a generic Notification Model based on the Java event model.
- Notifications can be emitted by MBean instances, as well as by the MBean server.
- By using a generic event type, this notification model allows any one listener to receive all types of events from a broadcaster.
- The filter is provided by the listener to specify only those events which are needed. Using a filter, a listener only needs to register once in order to receive all selected events of an MBean.
- Notification filters are usually implemented as callback methods of the listener itself.

Notification Model

JMX Architecture → Instrumentation Level → Notification Model

- The JMX notification model relies on the following components:
  - *A generic event type*, Notification, which can signal any type of management event.
  - The *NotificationListener interface*, which needs to be implemented by objects requesting to receive notifications sent by MBeans.
  - The *NotificationFilter interface*, which needs to be implemented by objects which act as a *notification filter*. This interface lets notification listeners provide a filter to be applied to notifications emitted by an MBean.
  - The *NotificationBroadcaster interface*, which needs to be implemented by each MBean wanting to emit *notifications*. This interface allows listeners to register their interest in the notifications emitted by an MBean.

MBean Meta Data Classes

JMX Architecture → Instrumentation Level → MBean Meta Data Classes

- The metadata classes contain the structures to describe all of the components of an MBean's management interface: its attributes, operations, notifications and constructors.

- These classes are used both for the introspection of standard MBeans and for the self-description of all dynamic MBeans.

- For each of these, the metadata includes a name, a description and its particular characteristics.

- One of the functions of the MBean server at the agent level is to provide the metadata of its MBeans.

- The different types of MBeans extend the metadata classes in order to provide additional information.

- The JMX agent exposes all of its MBeans, regardless of their type, through the MBean metadata classes.

Agent Level

JMX Architecture → Agent Level

- The agent level provides a specification for implementing agents. Management agents directly control the resources and make them available to remote management applications.

- This level builds upon and makes use of the instrumentation level, in order to define a standardized agent to manage JMX manageable resources.

- The *JMX Agent* consists of an MBean server and a set of services for handling MBeans.

- In addition, a JMX agent will need at least one communications adaptor or connector.

- Managers access an agent's MBeans and use the provided services through a protocol adaptor or connector.

MBean Server

JMX Architecture → Agent Level → MBean Server

- The Managed Bean server, or *MBean server* for short, is a registry for objects which are exposed to management operations in an agent.

- Any object registered with the MBean server becomes visible to management applications. However, the Mbean server only exposes an MBean's management interface, never its direct object reference.

- Any resource that you want to manage from outside the agent's Java virtual machine must be registered as an MBean in the server. The MBean server also provides a standardized interface for accessing MBeans within the same JVM, giving local objects all of the benefits of manipulating manageable resources.

- MBeans can be instantiated and registered by:
  - Another MBean
  - The agent itself
  - A remote management application (through the distributed services)

- The operations available on MBeans include:
  - Discovering the management interface of MBeans
  - Reading and writing their attribute values
  - Performing operations defined by the MBeans
  - Getting notifications emitted by MBeans
  - Querying MBeans based on their object name or their attribute values

Agent Services

JMX Architecture → Agent Level → Agent Services

- Agent services are objects that can perform management operations on the MBeans registered in the MBean server.

- The JMX specification defines the following agent services:
  - **Dynamic class loading** through the m-let (management applet) service retrieves and instantiates new classes and native libraries from an arbitrary network location.
  - **Monitors** observe an MBean attribute’s numerical or string value and can notify other objects of several types of changes in the target.
  - **Timers** provide a scheduling mechanism based on a one-time alarm-clock notification or on a repeated, periodic notification.
  - **The relation service** defines associations between MBeans and enforces the cardinality of the relation based on predefined relation types.

Distributed Services Level

JMX Architecture → Distributed Services Level

The distributed services level provides the interfaces for implementing JMX managers. This level defines management interfaces and components that can operate on agents or hierarchies of agents. These components can:

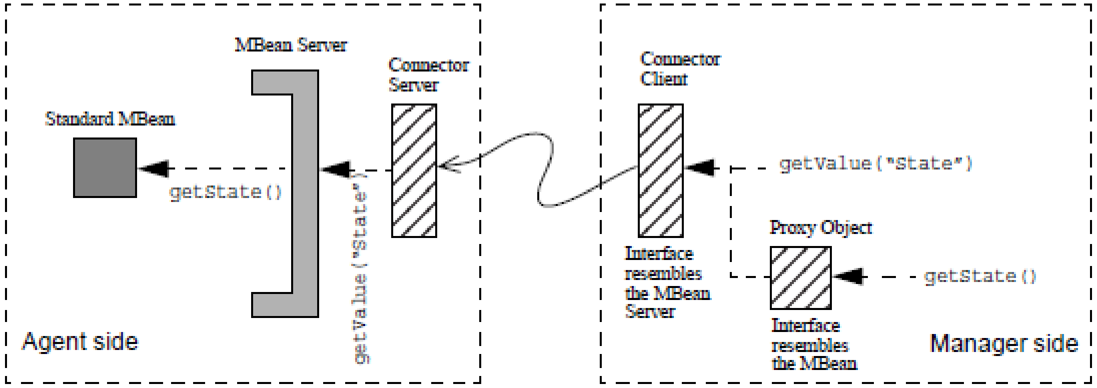
- Provide an interface for management applications to interact transparently with an agent and its JMX manageable resources through a connector.
- Expose a management view of a JMX agent and its MBeans by mapping their semantic meaning into the constructs of a data-rich protocol (for example HTML or SNMP).
- Distribute management information from high-level management platforms to numerous JMX agents.
- Consolidate management information coming from numerous JMX agents into logical views that are relevant to the end user's business operations.
- Provide security.

Protocol Adaptors and Connectors

- *Protocol adaptors* and *Connectors* make the agent accessible from remote management applications. They provide a view through a specific protocol of the Mbeans instantiated and registered in the MBean server. They enable a management application outside the JVM to:
  - Get or set attributes of existing MBeans
  - Perform operations on existing MBeans
  - Instantiate and register new MBeans
  - Register for and receive notifications emitted by MBeans

- Both connector servers and protocol adaptors use the services of the MBean server in order to apply the management operation they receive to the MBeans, and in order to forward notifications to the management application.

Propagation of a Remote Operation to an MBean



Protocol Adaptors and Connectors

- **Connectors** are used to connect an agent with a remote JMX-enabled management application; that is: a management application developed using the JMX distributed services. This kind of communication involves a connector server in the agent and a connector client in the manager.
- These components convey management operations transparently point-to-point over a specific protocol. The distributed services on the manager side provide a remote interface to the MBean server through which the management application can perform operations. A connector is specific to a given protocol, but the management application can use any connector indifferently since they have the same remote interface.

- **Protocol adaptors** provide a management view of the JMX agent through a given protocol. They adapt the operations of MBeans and the MBean server into a representation in the given protocol, and possibly into a different information model.
- Management applications that connect to a protocol adaptor are usually specific to the given protocol. This is typically the case for legacy management solutions that rely on a specific management protocol. They access the JMX agent not through a remote representation of the MBean server, but through operations that are mapped to those of the MBean server.

- A protocol connector (e.g. the JMX RMI Connector) exposes the MBeans as they are - so a remote client sees the same model than a local client.
  A protocol adaptor (e.g. an SNMP adaptor, or HTML adaptor) performs a model mediation - to adapt the model to what a client of that protocol (e.g. SNMP Manager, or Web Browser) would expect to see.

Why JMX

- **Enables Java applications to be managed without heavy investment**.
  A JMX technology-based agent (JMX agent) can run on most Java technology-enabled devices. Consequently, Java applications can become manageable with little impact on their design. A Java application needs only to embed a managed object server and make some of its functionality available as one or several managed beans (MBeans) registered in the object server. That is all it takes to benefit from the management infrastructure.

- **Provides a standard way to manage Java applications, systems, and networks**.

- **Can be used for out-of-the-box management of the Java VM**.
  The JVM is highly instrumented using the JMX technology. You can start a JMX agent to access the built-in Java VM instrumentation, and thereby monitor and manage a Java VM remotely.

- **Provides a scalable, dynamic management architecture**.
  Every JMX agent service is an independent module that can be plugged into the management agent, depending on the requirements. JMX solutions can scale from small-footprint devices to large telecommunications switches and beyond.

- **Leverages existing standard Java technologies**.

- **Integrates with existing management solutions and emerging technologies**.
  The JMX APIs are open interfaces that any management system vendor can implement. JMX solutions can use lookup and discovery services and protocols such as Jini network technology and the Service Location Protocol (SLP).

Epilogue

- The Java Management extensions technology brings unique facilities to such solutions: portability, on-demand deployment of management functionality, dynamic and mobility services, and security.

- If you are using JMX to gain insight into what a server application is doing, you need a means of identifying and tracking units of work

- To get the most value out of JMX, you may want to consider at design time what kinds of data would be useful to users and operators at run time.

- With the release of Java 1.5, JMX is built-in and readily usable in even the smallest of applications. It is now a simple task to instrument nearly any application and expose important monitoring metrics. Custom applications, involving many processes and multiple middleware components, may be effectively managed from a remote console.

- This frees the application developer from the burden of building management instrumentation. Application developer no longer need to manually add instrumentation code. The application developer can select the information and controls that need to be exposed for management.

THANK YOU