Ecole
« Intergiciel et Construction des Applications Réparties »

Nice - Maison des séminaires
25 août - 29 août 2008
Organisée par le laboratoire I3S
Java™ Management Extensions (JMX™)

Éamonn McManus
Sun Microsystems Inc
eamonn.mcmanus@sun.com
http://java.sun.com/jmx
http://weblogs.java.net/blog/emcmanus
Who am I?

- Specification Lead for JMX-related JSRs
  - JSR 160 (JMX Remote API)
  - JSR 255 (JMX API 2.0)
  - JSR 262 (Web Services Connector for JMX Agents)

- Technical Lead of Sun's JMX team since 2000
  - based in Grenoble, France
  - part of the JDK organization

- eamonn.mcmanus@sun.com
- http://weblogs.java.net/blog/emcmanus
Agenda

■ Introduction to JMX Technology
■ Aspect-Oriented Programming (AOP)
■ Evolution of the JMX API
■ Some areas of investigation
Agenda

- Introduction to JMX Technology
- Aspect-Oriented Programming (AOP)
- Evolution of the JMX API
- Some areas of investigation
JMX™ API

- A standard part of the Java Platform, Standard Edition (Java SE platform)
  - Starting with version 5.0
  - Also part of Java 2 Platform, Enterprise Edition (J2EE™) 1.4 platform

- Can be used for management...
  - For example, changing configuration settings

- ...and monitoring...
  - For example, obtaining statistics and error notifications

- ...of running applications
  - From big server applications to embedded device controllers
Example: VM Instrumentation

- As of version 5.0, the Java platform exports lots of monitoring data via JMX technology
  - Number of threads
  - Stack of any given thread
  - Sizes of different memory areas
  - Time spent doing garbage collection
  - Number of classes
  - Class path

- You can also cause a garbage collection remotely
JConsole: JVM overview tab
JConsole: JVM info as MBeans
JConsole: app server data as MBeans

![Image of JConsole interface showing app server data as MBeans]
Example: an online poker server

- You have a continuously-running poker server
  - of course players are not playing for *money*

- You want to be able to see...
  - how many players are currently connected
  - how many games are in progress
  - how many robot players are running
  - statistics on robot player performance

- You want to be *notified*...
  - if a player is refused a connection
  - if a player is winning suspiciously often

- You want to *control*...
  - maximum number of players who can connect
  - robot player parameters
Adding JMX Instrumentation to Your Application

Remote Manager

Application

JMX Agent

Manages
MBeans

- An MBean is a named **managed object** representing a resource
  - Application configuration setting
  - Program module
  - Device
  - Collection of statistics
  - etc.

- An MBean can have:
  - Attributes that can be read and/or written
  - Operations that can be invoked
  - Notifications that the MBean can send
MBean Example

PokerControlMBean

Attributes

- ConnectedPlayerCount: int  R
- MaxConnectedPlayers: int  RW

Operations

- ejectPlayer(name: String): void
- otherOperation: OtherType

Notifications

- “com.example.player.joined”
- “com.example.player.player.refused”
Standard MBeans

- There are several kinds of MBeans
  - Standard, MXBean, Dynamic, Model, Open
- The simplest are Standard MBeans and their cousins, MXBeans
- To make a Standard MBean:
  1. Write a bean interface called `SomethingMBean`
  2. Implement it in a class called `Something`
  3. Then an instance of `Something` is a Standard MBean
Standard MBean Example

```java
public interface PokerControlMBean {
    // a read-write attribute called MaxConnectedPlayers
    // of type int
    public int getMaxConnectedPlayers();
    public void setMaxConnectedPlayers(int n);

    // a read-only attribute called ConnectedPlayerCount
    // of type int
    public int getConnectedPlayerCount();

    // an operation called ejectPlayer
    // with a String parameter
    public void ejectPlayer(String name);
}

public class PokerControl implements PokerControlMBean {
    public int getMaxConnectedPlayers() {
        ...logic to determine MaxConnectedPlayers...
    }
}
```
A variant of Standard MBeans introduced in the Java SE 6 platform

Basically the same as Standard MBeans when all attribute and operation types are “simple”

```java
public interface PokerControlMXBean {
    public int getConnectedPlayerCount();
    public void ejectPlayer(String name);
}
```

User-defined JavaBean classes mapped to a set of standard types

```java
public interface PokerControlMXBean {
    public PokerStats getCacheStats();
}
```

Clients (like JConsole) don’t need to know PokerStats, only the standard types
public interface PokerControlMXBean {
    // a read-write attribute called MaxConnectedPlayers
    // of type int
    public int getMaxConnectedPlayers();
    public void setMaxConnectedPlayers(int n);

    // a read-only attribute called ConnectedPlayerCountCode
    // of type int
    public int getConnectedPlayerCount();

    // an operation called ejectPlayer
    // with a String parameter
    public void ejectPlayer(String name);
}

public class PokerControl implements PokerControlMXBean {
    public int getMaxConnectedPlayers() {
        ...logic to determine MaxConnectedPlayers...
    }
    ...
MBean Server (JMX Agent)

com.example:type=PokerControl
Registering an MBean

public class PokerControl implements PokerControlMBean
{
...

MBeanServer mbs =
    ManagementFactory.getPlatformMBeanServer();

PokerControl mbean = new PokerControl();
ObjectName name =
    new ObjectName("com.example:type=PokerControl");

mbs.registerMBean(mbean, name);

- See online tutorial, or documentation for package
  javax.management

- Everything can be found from:
  http://java.sun.com/jmx
Making the JMX Agent Connectable

- **Java Development Kit (JDK) version 1.4**
  - Add JMX API jars to the classpath
  - Use JMX remote API to set up connectivity

- **JDK version 5**
  - Run your app with system properties such as
    - `-Dcom.sun.management.jmxremote`

- **JDK version 6**
  - It just works!
  - (If you are connecting from the same machine)

- You can always use JConsole from a later JDK version to connect to an agent on an earlier one
Notifications

- JMX MBeans can emit notifications when things happen
- Clients can “listen” for notifications from an MBean
  - both local and remote clients
- When the MBean emits a notification, all of its listeners get told
How an MBean emits notifications

- The MBean must implement NotificationBroadcaster, usually by subclassing or delegating to NotificationBroadcasterSupport

```java
public class PokerControl
    extends NotificationBroadcasterSupport
    implements PokerControlMBean {
    ...
    Notification n = new Notification(...);
    sendNotification(n);
    ...
}
How a client listens for notifications

- A client adds a listener through the MBean Server
  - object implementing NotificationListener
  - its handleNotification method is invoked with each notification
  - this works even if the MBean Server is remote

```java
public class React implements NotificationListener {
    public void handleNotification(Notification n, Object h) {
        if (n.getType().equals("poker.cheater.detected"))
            handleCheater(n);
    }
}

NotificationListener listener = new React(...);

mbeanServer.addNotificationListener(
    pokerControlName, listener, null, null);
```
Agenda

- Introduction to JMX Technology
- Aspect-Oriented Programming (AOP)
- Evolution of the JMX API
- Some areas of investigation
Aspect-Oriented Programming (AOP) allows you to cause extra code to be executed at certain points in your program

- “Every time my code calls File.createTempFile, add one to a counter”
- “Every time my code gets an exception, log a message”
- “Every time a class in com.example.mypackage calls a method in java.net, print a stack trace”
- “Every time anyone calls a public method in Foo class, check they have FooPermission”
AOP Vocabulary

- AOP terminology seems to have been designed to be as cryptic as possible
  - “join points”, “pointcuts”, “advice”, “aspects”
- I will explain the jargon where relevant, then avoid using it
Pointcuts

- AOP gives you a language to describe points of interest in your program
  - “A call to File.createTempFile”
  - “A call to any method in java.io.File”
  - “A call to File.createTempFile from code in com.example.mypackage and where the second argument is not null”
  - “Construction of any object from within the class com.example.mypackage.MyClass”

- These are pointcuts
  - A pointcut is a set of join points, but you don't care
Once you have specified what points in the execution of the program are of interest, you can say what you want to happen:

- **before**
- **after**
- **around**

“Before every call to File.createTempFile, increment this counter”

“Around every call to Socket.connect, measure the time taken”

This is called *advice*, for some reason
Proceed With Caution
Evil AOP

Warning: subjectivity

- You can inject *any* code into *any* method in your application.
- This injected code can *change* what the method does.
- So what you see is no longer what you get.
- This can destroy one of the big advantages of the Java programming language, its transparency.
- We will describe ways to inject code that *observes* what is going on, but does not change it.
AspectJ and @AspectJ (1)

- AspectJ is an extension of the Java programming language that adds AOP constructs

```java
before(): call(* java.io.File.createTempFile(..)) {
    tempFileStats.addCall();
}
```

- @AspectJ is a set of annotations you can use in a normal Java platform program to the same effect

```java
@Before("call(* java.io.File.createTempFile(..))")
public void updateTempFileStats() {
    tempFileStats.addCall();
}
```
AspectJ and @AspectJ (2)

- AspectJ requires a special compiler (ajc)
- @AspectJ can be used with the standard compiler (javac)
- @AspectJ does its work when classes are loaded at runtime
- With @AspectJ, most syntax errors are only detected at runtime
Using AOP to Add Probes to Code

```java
@Aspect
public class MyAspect {
    static TempFileStats tempFileStats = new TempFileStats();

    @Before("call(* java.io.File.createTempFile(..))")
    public void updateTempFileStats() {
        tempFileStats.addCall();
    }
}
```
Using AOP to Add Probes to Code

```java
public void saveConfig() {
    String config = ...;

    tempFileStats.addCall();

    File tmp = File.createTempFile("config", null);
    OutputStream out = new FileOutputStream(tmp);
    ...
}
```
Glassbox

- An Open-Source project based on Spring
  - www.glassbox.com

- Uses AspectJ to insert timing measurements into arbitrary apps at runtime
  - no code modification needed

- Exposes measurement results as MBeans

- A special web app analyzes these results and reports on possible problems
Glassbox MBeans
Glassbox Analysis
Agenda

■ Introduction to JMX Technology
■ Aspect-Oriented Programming (AOP)
■ Evolution of the JMX API
  ◆ Web-Services Connector
  ◆ Namespaces
  ◆ Event Service
  ◆ Miscellaneous
■ Some areas of investigation
Evolution of the JMX API

- Two Java Specification Requests (JSRs) in progress
- JSR 255 is defining version 2.0 of the JMX API
- JSR 262 is defining a Web Services Connector for JMX Agents
- We expect that the Java Platform, Standard Edition (Java™ SE Platform), version 7, will include both JSRs
Agenda

■ Introduction to JMX Technology
■ Aspect-Oriented Programming (AOP)
■ Evolution of the JMX API
  ◆ Web-Services Connector
  ◆ Namespaces
  ◆ Event Service
  ◆ Miscellaneous
■ Some areas of investigation
Strong requirements expressed by our customers:

- Integrate JMX technology in http/web architectures
- Allow JMX technology to interoperate outside of the Java world
I am a developer using the JMX API, why should I care about WS-Connector?

- Java platform clients can use JMX Remote API
  - Fits into JMX Remote API Open Architecture
  - Protocol choice is nearly transparent
  - Just a matter of changing a URL:
    service:jmx:ws://<host>:<port>/<http context>
  - JMX API's simplicity and dynamicity retained

- Take advantage of intrinsic Web Services properties:
  - Fit into web infrastructure
  - Firewall friendliness
  - Interoperability with non Java technology-based client
  - Standard interoperability based on WS-* standards
A Connector based on Standards

- Defined by JSR 262 (Public Review, standard in progress)
  - Candidate for Java SE 7

- Based on WS-Management (DMTF)
  - http://www.dmtf.org/standards/wsman/

WS-Management Request

Client → Request → JSR 262 Server → Response → MBean Server

WS-Management Response
WS-Management (WS-Man) standard

- General SOAP-based protocol (SOAP 1.2)
- For managing systems (Servers, Devices, Applications, Services, ...)
- Relies on existing set of Web service specifications (WS-*)
- Adoption is growing:
  - Windows Vista, Windows XP SP2, Windows 2003 Server,...
  - Building block for Management of Virtualization:
    - Sun (xVM Server), Microsoft, VMWare, Novell,...
    - DMTF Systems Management Architecture for Server Hardware (SMASH) 2.0
WS-Man to manage Resources

- Identified by a WS-A endpoint reference
- Associated with an XML schema (XSD)
- Has a type: `<wsman:resourceURI>`
- Multiple instances of the same type: `<wsman:Selector>`
- Has values
- Has operations
- Can emit notifications
JMX specification to manage MBean

- Identified by an ObjectName
- Associated with an MBeanInfo
- Has a type: `javax.management.ObjectName`
- Multiple instances of the same type: ObjectName keys
- Has values: MBean Attributes
- Has operations: MBean Operations
- Can emit notifications: MBean Notifications
Web-Services Connector to map MBean to Resource

- Automatic conversion
- Select your interaction path
Web-Services Connector to map MBean to Resource

- Automatic conversion
- Select your interaction path
Web-Services Connector to map MBean to Resource

- Automatic conversion
- Select your interaction path
Web-Services Connector to map MBean to Resource

- Automatic conversion
- Select your interaction path

WS Connector Server

MBeanServer

JMX

WS-Man
Non Java client

WinRM, ViPerl, ...

Automatic conversion
Select your interaction path
Web-Services Connector to map MBean to Resource

- Automatic conversion
- Select your interaction path

[Diagram]

- Automatic conversion
- Select your interaction path

[Diagram]

- Automatic conversion
- Select your interaction path

[Diagram]
Web-Services Connector to map MBean to Resource

- Automatic conversion
- Select your interaction path

![Diagram showing MBeanServer, WS Connector Server, WS Connector Client, JMX, JMX client, JConsole, Visual VM, WinRM, ViPerl]
WS-Man View of a getAttribute, the request

Boolean \( b = (\text{Boolean}) \text{connection.getAttribute}(\text{new ObjectName("java.lang:type=Memory"), "Verbose"}); \)

<env:Envelope>
  <env:Header>
  </env:Header>
  <wsman:SelectorSet>
    <wsman:Selector Name="ObjectName">
      java.lang:type=Memory
    </wsman:Selector>
  </wsman:SelectorSet>
  <wsman:ResourceURI>
    http://jsr262.dev.java.net/DynamicMBeanResource
  </wsman:ResourceURI>
  <wsman:FragmentTransfer>
    //jmx:Property[@name="Verbose"]
  </wsman:FragmentTransfer>
</env:Envelope>
Boolean \( b = (\text{Boolean}) \text{connection.getAttribute(new ObjectName("java.lang:type=Memory"), "Verbose"}); \)

```xml
<env:Envelope>
  <env:Header>
    <wsman:SelectorSet>
      <wsman:Selector Name="ObjectName">
        java.lang:type=Memory
      </wsman:Selector>
    </wsman:SelectorSet>
    <wsman:ResourceURI>
      http://jsr262.dev.java.net/DynamicMBeanResource
    </wsman:ResourceURI>
    <wsman:FragmentTransfer>
      //jmx:Property[@name="Verbose"]
    </wsman:FragmentTransfer>
  </env:Header>
</env:Envelope>
```
WS-Man View of a getAttribute, the request

Boolean \( b = (\text{Boolean}) \text{connection.getAttribute(new ObjectName("java.lang:type=Memory"), "Verbose"}); \)

<env:Envelope>
  <env:Header>
  </env:Header>
  <wsman:SelectorSet>
    <wsman:Selector Name="ObjectName">
      java.lang:type=Memory
    </wsman:Selector>
  </wsman:SelectorSet>
  <wsman:ResourceURI>
    http://jsr262.dev.java.net/DynamicMBeanResource
  </wsman:ResourceURI>
  <wsman:FragmentTransfer>
    //jmx:Property[@name="Verbose"]
  </wsman:FragmentTransfer>
</env:Envelope>
WS-Man View of a getAttribute, the request

```java
Boolean b = (Boolean) connection.getAttribute(new ObjectName("java.lang:type=Memory"), "Verbose");
```

```xml
<env:Envelope>
  <env:Header>
  </env:Header>
  <wsman:SelectorSet>
    <wsman:Selector Name="ObjectName">java.lang:type=Memory</wsman:Selector>
  </wsman:SelectorSet>
  <wsman:ResourceURI>http://jsr262.dev.java.net/DynamicMBeanResource</wsman:ResourceURI>
  <wsman:FragmentTransfer>//jmx:Property[@name="Verbose"]</wsman:FragmentTransfer>
</env:Envelope>
```
WS-Man View of a getAttribute, the response

<env:Envelope>

...

<env:Body>

<wsman:XmlFragment>

<jmx:Property name="Verbose">
  <jmx:Boolean>false</jmx:Boolean>
</jmx:Property>

</wsman:XmlFragment>

</env:Body>

<env:Envelope>
MBean WS-Man Resource

- MBeans are mapped to a single generic WS-Man XML resource type

- WS-Man resource contains MBean Attributes
  
  ```xml
  <xs:element name="DynamicMBeanResource">
    <xs:complexType>
      <xs:sequence>
        <!-- The list of MBean attributes. Property composed of a Name and a Value -->
        <xs:element ref="Property" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  ```

- Single “invoke” operation to convey all MBean operations
JSR 262 standard JMX concepts to XML mapping

- MBean
- Attribute
- Operation
- Notification
- Notification Filter
- MBeanInfo
- Relations
- Query
- ObjectName
- ObjectInstance
- OpenType, CompositeData, TabularData
- JMXServiceURL
JSR 262 standard Java client to XML mapping

- Defines an XML representation for Java types exposed in MBean interfaces
  - Primitive and boxed types
  - Other Java™ types (URL, QName, etc.)
  - JMX OpenType, ObjectName, JMXServiceURL, ...
  - Some Collections, such as List, Map, Vector of above types
  - Unbounded Array of above types
  - Schema extensibility for custom types

- Defines a mapping from Exceptions into WS-Management faults
  - e.g. JMX InstanceNotFoundException ==> wsman:EndpointUnavailableFault
  - Mapping defined for Exception fault cause and stack trace elements
JSR 262 standard protocol mapping

- Mapping from JMX connector operations to WS-Management operations
  - Notification subscription and delivery
    - Pull and Push delivery modes
  - Get / set MBean attributes
  - Invoke MBean operations
  - MBean Metadata retrieval
  - MBeanServer queries

- Connected protocol
  - Comply with JSR 160 defined server and client sides connection life cycle: OPEN / CLOSE / FAIL
No new security model, relies on existing techniques

Authentication
- HTTP Basic Auth

Encryption
- HTTPS

Authorization
- Permissions from the JMX API

Plug your own model
- WS-Security can be used
Key points to remember

- Java JMX API clients please forget about the XML mapping!
- Usage as simple as RMI Connector.
- Tailored for MXBean
- Secure
- Strong interoperability
- Automatic exposure of MBeans as WS-Management resources
Reference Implementation details

- Early Access 3 downloadable from java.net
  - http://ws-jmx-connector.dev.java.net
- Runs on Java SE 5 and Java SE 6
- JAX-WS 2.1 Endpoint
- Exposes an API to adapt MBean representation to custom or standard (e.g. WS-CIM) XML definitions
- Proven Interoperability with Microsoft WinRM
## Interoperability Table

<table>
<thead>
<tr>
<th>WS-Man Client toolkit</th>
<th>Interop</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>WiseMan</td>
<td>OK</td>
<td>Deep testing</td>
</tr>
<tr>
<td>WinRM CLI/VBScript</td>
<td>OK</td>
<td>Deep testing</td>
</tr>
<tr>
<td>VMWare VIPerl</td>
<td>OK</td>
<td>Light testing</td>
</tr>
<tr>
<td>Openwsman</td>
<td>TODO</td>
<td></td>
</tr>
<tr>
<td>Intel® DTK</td>
<td>TODO</td>
<td></td>
</tr>
</tbody>
</table>
Agenda

- Introduction to JMX Technology
- Aspect-Oriented Programming (AOP)
- Evolution of the JMX API
  - Web-Services Connector
  - Namespaces
  - Event Service
  - Miscellaneous
- Some areas of investigation
Namespaces

MBeanServer

foo://com.other:type=DamageControl

bar://com.bug:type=PestControl

com.example:type=PokerControl

foo://
Namespaces

- Classic ObjectName looks like this:
  com.example:type=PokerControl

- Now ObjectNames can be organized into namespaces:
  pokerserver//com.example:type=PokerControl
  remote//pokerserver//com.example:type=PokerControl

- Each namespace is an object that implements the MBeanServer interface
Use case: several apps in the same VM

- The JMX API has always allowed having more than one MBean Server
  - `MBeanServerFactory.newMBeanServer()`
- For example, if you are running more than one app in the same VM
- With namespaces, you can group these MBean Servers into a higher-level MBean Server
Use case: cascading (federation)

- The MBeanServer for a namespace can be remote
Why cascading?

- Cascading provides a single point of access to a distributed set of MBean Servers
- Clients do not need to know how to find those MBean Servers
  - and may not be able to access them even if they do know
- Example: clustered app server
- Example: blade server
Cascading issues

Security: If I connect to the Master Agent, what permissions do I need to connect to a subagent?

- Master Agent can propagate my credentials to the subagent
- Or subagent can trust Master Agent to tell it who I am
- Or permission to connect to Master Agent suffices

Network failures: If the connection to the subagent breaks, what do I see in the Master Agent?

- “Phantom” namespace remains
- Accesses to phantom MBeans produce errors
- Can reconnect when possible again
Use case: Virtual MBeans

- The standard implementation of the MBeanServer interface has one Java object for every MBean
  - what if there is a huge number of them?
  - what if they come and go very fast?

- A different implementation can create the MBean at the exact moment it is accessed, then discard it

getAttribute("process//com.foo:type=Process,id=325", "Owner")

grep -r "Owner" /proc/325

dumpxml -a /proc/325
Client contexts

- A special namespace `jmx.context//` lets clients communicate context to MBeans
  - locale, transaction id, ...
  - but note that there is no explicit support for transactions

- A call to
  ```
  getAttribute("jmx.context//locale=fr//com.example:type=Foo", "Bar")
  ```
becomes a call to
  ```
  getAttribute("com.example:type=Foo", "Bar")
  ```
with "locale=fr" available in the thread

- An MBean can return localized strings in its attributes
- It can also localize the descriptions in its MBeanInfo
Client locales

- com.example.PokerMain (pid 5936)
  - MBeans
    - Attributes
      - Name: MaxPlayerCount, Value: 50
      - Name: PlayerCount, Value: 15
    - Number of players currently connected

- Nombre de joueurs actuellement connectés
  - Name: MaxPlayerCount, Value: 50
  - Name: PlayerCount, Value: 15
Agenda

- Introduction to JMX Technology
- Aspect-Oriented Programming (AOP)
- Evolution of the JMX API
  - Web-Services Connector
  - Namespaces
  - Event Service
  - Miscellaneous
- Some areas of investigation
■ JMX Remote API (JSR 160) allows notifications to be received remotely

■ Very loose coupling between client and server has good properties when clients are numerous, but has bad properties too:
  ◆ When there are many notifications, a client can miss some, even if it is only interested in very few
  ◆ An MBean does not know who is listening to it, so cannot adjust its behavior per client
  ◆ Difficult to impose fine-grained security

■ Event Service fixes these problems, without changing the client/server protocol

■ Also allows listening to a set of MBeans in one operation
Event Service: custom transports

- Event Service lets you use custom transports such as JMS, e-mail, JavaSpaces, etc
  - only “RMI push” supplied as standard
Agenda

■ Introduction to JMX Technology
■ Aspect-Oriented Programming (AOP)
■ Evolution of the JMX API
  ◆ Web-Services Connector
  ◆ Namespaces
  ◆ Event Service
  ◆ Miscellaneous
■ Some areas of investigation
Defining MBeans with annotations

```java
public interface PokerControlMBean {
    int getPlayerCount();
    void eject(String name);
}

public class PokerControl implements PokerControlMBean {

    public int getPlayerCount() {
        return something;
    }

    public void eject(String n) {
        doSomething(name);
    }
}

@MBean
public class PokerControl {
    implements PokerControlMBean {

        public int getPlayerCount() {
            return something;
        }

        public void eject(String n) {
            doSomething(name);
        }
    }
}
Annotations for descriptions

![Diagram of annotations for descriptions]

- **com.example.PokerMain (pid 3266)**
  - **Attributes**
    - **MaxPlayerCount**: 50
    - **PlayerCount**: 15
  - **Number of players currently connected**

*Image and diagram design © 2007, M. Riveill*
Annotations for descriptions

```java
@Description("Control the poker server")
public interface PokerControlMBean {
    @Description("Number of players currently connected")
    public int getPlayerCount();

    @Description("Maximum number of players that can connect")
    public int getMaxPlayerCount();
    public void setMaxPlayerCount(int max);

    @Description("Disconnect a named player")
    public void eject(@Description("Name") String name);

    @Description(value="Shut down the server", key="shut.down")
    public void shutdown();
}
```
JMX API has always had queries to find MBeans matching certain criteria

Queries inspired by SQL but must be constructed with code:

- QueryExp query = Query.and(
  Query.gt(Query.attr("ConnectTime"),
  Query.value(60)),
  Query.eq(Query.attr("Type"),
  Query.value("Newbie")));

New SQL-like language:

- QueryExp query = Query.fromString(
  "ConnectTime > 60 and Type = 'Newbie'");

Code much easier to write and read

Provides a simple way for clients like Java™ VisualVM to input user queries

The idea of a JMX query language was first proposed by Norbert Lataille & Marc Fleury in 2000; the language here is not derived from that proposal.
User-defined MXBean mappings

■ Sometimes the predefined MXBean mapping rules aren't suitable for your custom types
  ◆ self-referential types not supported
  ◆ subclassing not supported

■ New classes and annotations allow you to extend the rules

```java
public class MyLinkedListMapping extends MXBeanMapping {
  ...
}

@MXBeanMappingClass(MyLinkedListMapping.class)
public class MyLinkedList {
  ...
  public MyLinkedList getNext() { ... }
  ...
}
```
Agenda

■ Introduction to JMX Technology
■ Aspect-Oriented Programming (AOP)
■ Evolution of the JMX API
■ Some areas of investigation
Aggregation

In big systems, often you want to aggregate data from several objects

- example: Poker Server running on a cluster, want to see data aggregated from the different Java VMs in the cluster
  - min, max, average response time

JMX API could provide some support for this?

- needs to be generic enough to be useful in a broad variety of cases
- needs to be specific enough that it is simpler than just doing it yourself
**Historical data**

- JMX technology allows you to see the instantaneous state of your MBeans.

- Often you want to see how this state has evolved:
  - is the response time getting longer?
  - are there peak times in the day?

- Currently the easiest way to track this is for the client to request and store the data periodically:
  - means extra work in the server and extra network traffic.

- Could the server have a way to store the data, and export a summary?
  - perhaps with an interface to drill down.
JMX API in Java EE platform

- JSR 77 specifies some standard MBeans that every app server must export
  - generic information about the app server itself
  - deployed apps and servlets; stats for JDBC, JMS, etc; server instances

- It does not specify a standard way to connect to the MBean Server
  - predates JSR 160 (JMX Remote API)

- It does not specify a standard way for an app to find the MBean Server and put its own MBeans there
Linking OSGi and JMX technologies

- MOSGi project deploys a JMX MBean Server as a service within OSGi, and provides a way for any OSGi bundle to register MBeans
- This is the opposite of the situation with Java EE!
  - We have a way for “apps” (bundles) to define their own management
  - But we don't have a generic management API for the OSGi platform itself
  - We’d like to be able to see and control the platform remotely via JMX protocols
    - see what bundles are deployed and their dependencies
    - ask a bundle where it gets some specific class from
    - perhaps remotely download a new bundle and update other bundles to reference the new version?
Ideas for the afternoon session

- We can discuss these ideas and any others at this afternoon's free-form session
More to explore

- **JMX home page:**
  - [http://java.sun.com/jmx](http://java.sun.com/jmx)
  - links to downloads, forums, articles, books, and more

- **My blog:**
  - [http://weblogs.java.net/blog/emcmanus](http://weblogs.java.net/blog/emcmanus)
  - Also see the link “Blogs” on the JMX home page

- **Visual VM:**
  - [http://visualvm.dev.java.net/](http://visualvm.dev.java.net/)