What can JMX do for you?

Simon Brown

Detica
INFORMATION INTELLIGENCE

Goals

- To give you the basics of JMX
- To show some of the use cases where JMX is applicable
- To get you infected so that you leave here and add JMX support to your own applications

Detica

Agenda

- 1. What is JMX?
- 2. How do you start using JMX?
- 3. Enhanced monitoring with notifications
- 4. Remote JMX
- 5. Use cases for enterprise applications
- 6. Programmatic access to MBeans
- 7. JMX and Spring
- 8. Extending JConsole
- 9. Case study
- 10. Summary

Detica

1. What is JMX?

Detica

Java 5 adoption

- Java 5, Standard Edition is a recent switch for many project teams but it's certainly mainstream now
- New features used include
  - Generics
  - Enhanced for loop
  - Autoboxing
- JMX, although now built-in to the platform isn't on that list
  - A real hidden gem but nobody seems to use it
  - JMX is very powerful yet simple and incredibly useful

Detica

You'll become infected

- I liken JMX to JUnit
- It sounds really useful
- It's a "nice to have"
- Once you try it, you get test infected
- You'll want to start testing all of your components
- The same goes for monitoring and managing your components

Detica

Monitoring

- “To keep track of systematically with a view to collecting information”
- “To keep close watch over; supervise”
- “The act of observing something (and sometimes keeping a record of it)”

Detica

Management

- “The act or manner of managing; handling, direction, or control”
- “The act, manner, or practice of managing; handling, supervision, or control”
- “Executive ability”

Detica

What can JMX do for you?

- JMX provides the infrastructure to support monitoring and management of your Java applications

Detica

Monitoring with JMX

- You can peak at the components running inside a JVM
- You can look at their attributes
- You can subscribe to any notifications they broadcast
- You can be alerted whenever something happens (e.g. an attribute passes a threshold)

Detica

Management with JMX

- You can invoke operations on the components running inside a JVM
  - Change their state
  - Change their behaviour
  - Change their configuration
  - Stop them
  - Start them

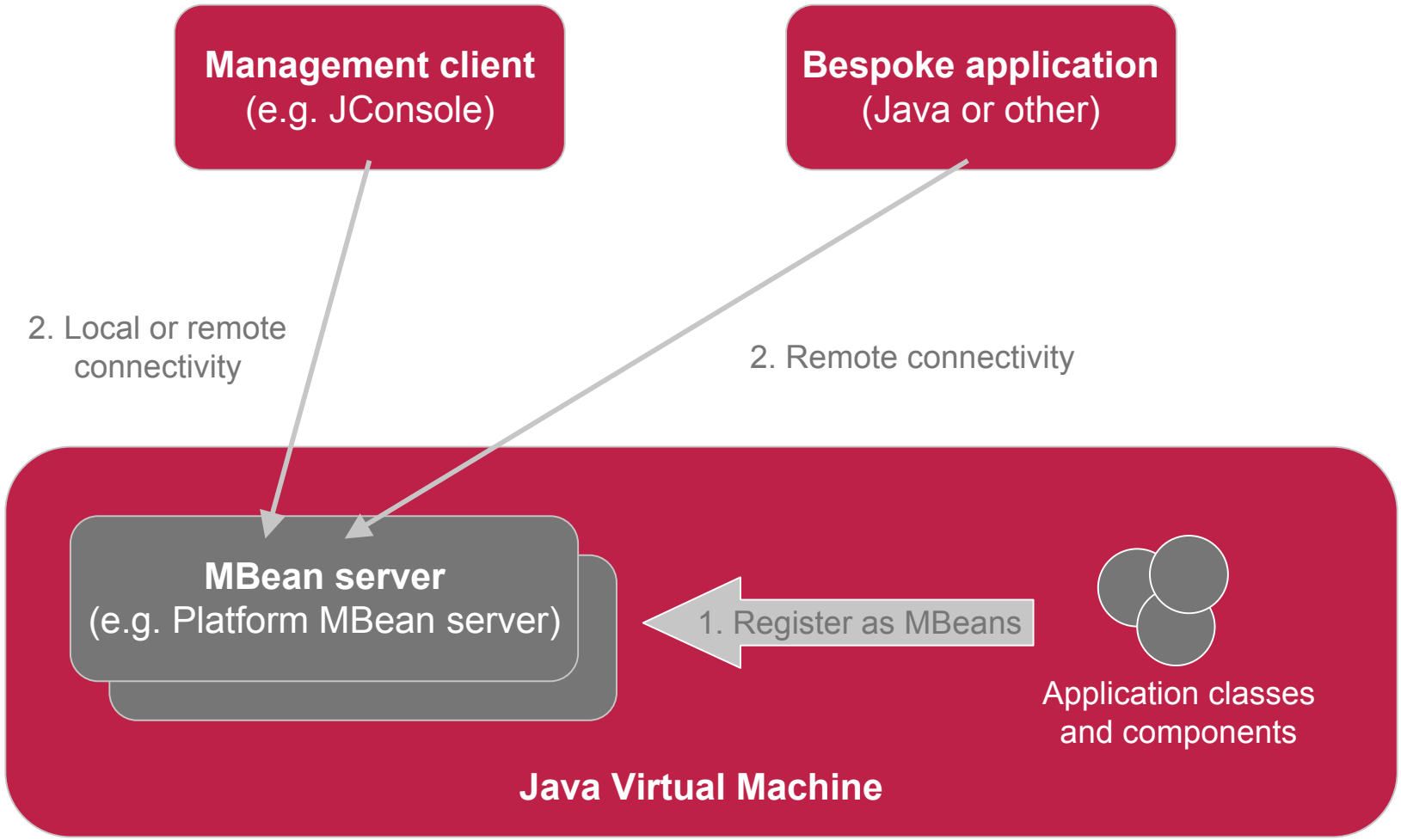
Detica

How do you do this?

- Inside the JVM are
  - Your application classes and components
  - One or more MBean (managed bean/resource) servers
- Your application registers the components it is interested in monitoring and managing with the MBean server

Detica

How do you do this?

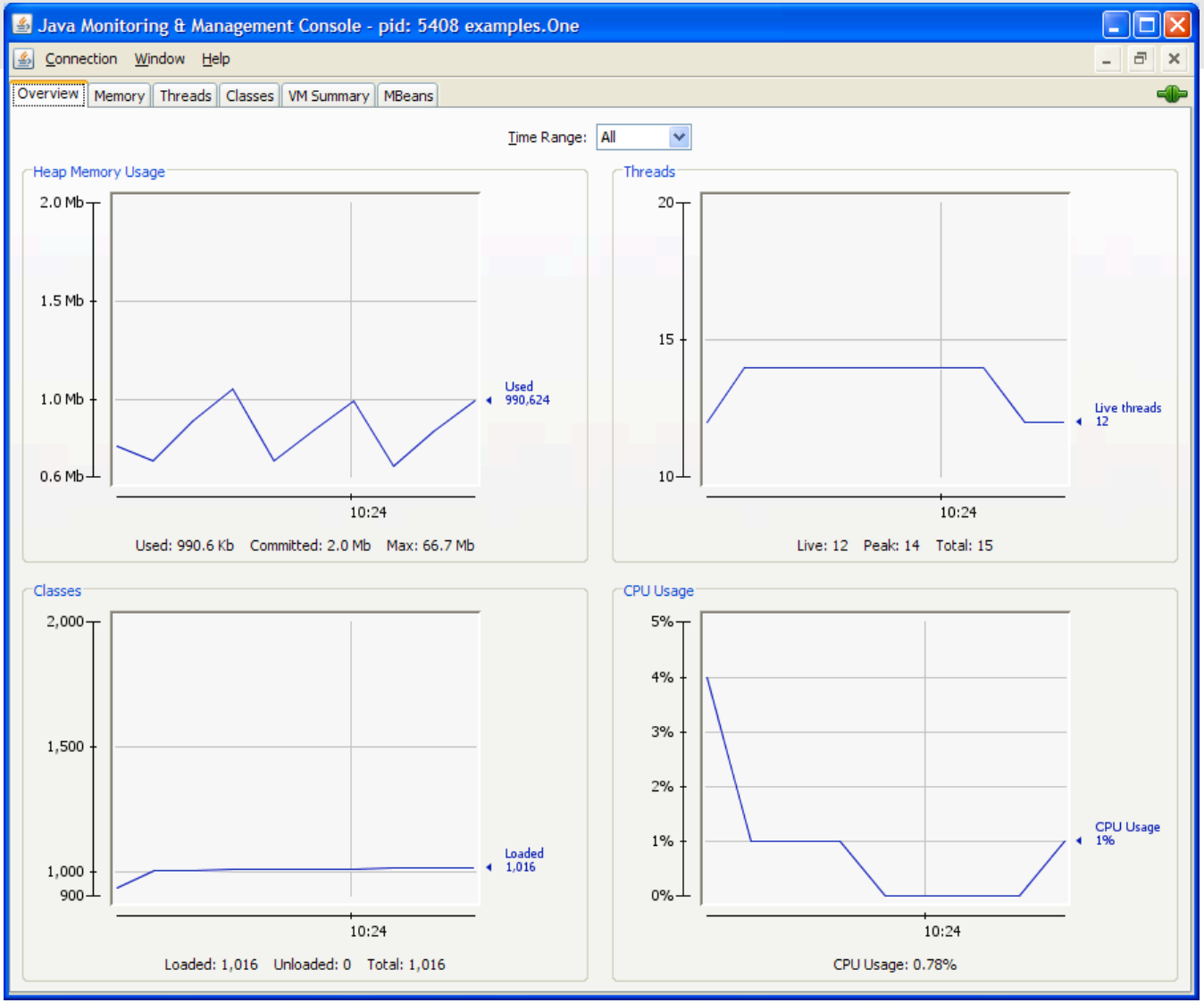


Detica

Demo 1 : Monitoring a JVM

- Java 6
  - JMX agent enabled by default via Dynamic Attach
- Java 5
  - you need to use a system property when starting the JVM
  - -Dcom.sun.management.jmxremote

Detica


Java Monitoring & Management Console - pid: 5408 examples.One
Connection
Window
Help
Overview
Memory
Threads
Classes
VM Summary
MBeans
Time Range:
All
Heap Memory Usage
2.0 Mb
1.5 Mb
1.0 Mb
0.6 Mb
Used
990,624
10:24
Used: 990.6 Kb
Committed: 2.0 Mb
Max: 66.7 Mb
Threads
20
15
10
Live threads
12
10:24
Live: 12
Peak: 14
Total: 15
Classes
2,000
1,500
1,000
900
Loaded
1,016
10:24
Loaded: 1,016
Unloaded: 0
Total: 1,016
CPU Usage
5%
4%
3%
2%
1%
0%
CPU Usage
1%
10:24
CPU Usage: 0.78%

Detica

2. How do you start using JMX?

Detica

Sample application

- Standalone JVM
- Consists of a number of components, one of which is a message consumer
- On application startup, the consumer connects to a message source and starts consuming messages
- It logs some information each time a message is received and consumed

Detica

Message consumer

```
public class MessageConsumer implements Runnable {
    public synchronized void start() { … }
    public void run() { … consume messages … }
    public synchronized void stop() { … }
    public boolean isRunning() { … }
    public long getMessagesReceived() { … }
    public long getMessagesProcessed() { … }
    public Message[] getRecentMessages() { … }
}
```

Detica

Application

```
public static void main(String[] args) {
   MessageConsumer consumer =
         new MessageConsumer();
   consumer.start();

   MessageProducer producer = new MessageProducer();
   }
```

Monitoring an application

- How would you ensure an application is running?
  - Windows Task Manager
  - *NIX ps
  - Java 6 jps
  - Monitor stdout
  - tail –f <logfile>
  - Check messages are being consumed

Detica

Or you could use JMX

- Use JMX to instrument the message consumer component
- Instrument : enhancing the implementation to expose information for monitoring and behaviour for management

Detica

MBeans

- There are several different ways to instrument a Java component
- Each component is a “managed resource” or “managed bean”
- There are several different types of MBeans
  - Standard MBeans and MXBeans
  - Dynamic MBeans
  - Open MBeans

Detica

Standard MBeans

- Easiest to implement
- Will allow you to cover the majority of your monitoring and management use cases
- The other types of MBeans are a little more complicated, although they have their uses

Detica

What is a Standard MBean?

- It’s a simple managed bean that is implemented through the use of
  - A Java interface
  - The JavaBeans naming conventions
  - Some additional naming conventions for JMX

Detica

What is a Standard MBean?

- We have a Java class called MessageConsumer
- How do we make this a Standard MBean?

Detica

Define the management interface

- Extract the attributes and operations that you wish to expose into an interface
- This is called the “management interface”
- JMX conventions must be followed
  - Interface must be named <Class name>MBean
  - Interface must be in the same package as the implementation

Detica

What gets exposed?

- The management interface allows you to expose attributes and operations
- Attributes are the properties of your MBean and you can get/set these at runtime
- Operations are arbitrary methods that you can call at runtime

Detica

Exposing attributes

- Attributes follow the normal JavaBeans naming conventions
  - Getters : getX() and isX()
  - Setters : setX()
- Attributes can be
  - Read-Write (getter and setter exposed)
  - Read only (only getter exposed)
  - Write only (only setter exposed

Detica

Exposing attributes

- Basic types
  - Primitives and their wrappers
  - String
  - BigDecimal
  - BigInteger
  - Date
- Arrays and collections of basic types

Detica

Exposing operations

- Operations are the other methods in your management interface that don’t meet the naming conventions for attributes
  - start()
  - stop()

Detica

Is that it?

- We still need to tell the JVM that we want to monitor and manage the MessageConsumer instance
- We need to tell the JVM that the MessageConsumer instance is a managed bean
- We need to register the managed bean with the MBean server, inside the JVM

Detica

Object names

- Each MBean is registered with a unique object name
  - Domain name (optional)
  - One or more unordered properties (name=value pairs)
- [domain name]:property1[,properties]

Detica

Object name conventions

- 1. specify a domain name to ensure uniqueness between components, particularly from different vendors/applications
- 2. specify a “type” property to reflect the type of the managed resource
- Examples
  - DefaultDomain:type=logmanager
  - com.company.application:type=MessageConsumer
  - com.company.application:type=MessageConsumer,instance=1
  - com.company.application:type=MessageConsumer,instance=2

Detica

Object name conventions

- An MBean server can be queried for matching MBeans and the naming conventions can make this easy
  - Find everything in a specific domain
  - Find everything of a specific type
  - Find everything a specific instance of MBean type X in domain Y
- We'll be looking at this after the break

Detica

Locating the MBean Server

- A JVM can contain any number of MBean servers
  - Platform MBean server
  - Application specific MBean servers
- ManagementFactory.getPlatformMBeanServer()
  - From the java.lang.management package
- MBeanServer interface
  - From the javax.management package

Detica

Registration

- Register components using the MBeanServer.registerMBean(Object, ObjectName) method
- Exceptional scenarios include
  - Non-unique object name (instance already exists)
  - Object isn't a compliant MBean

Detica

Registering an MBean instance

```
MessageConsumer consumer = new MessageConsumer();
try {
        ObjectName objectName = new ObjectName(
                "com.company.application:type=MessageConsumer");

        MBeanServer mbeanServer =
                ManagementFactory.getPlatformMBeanServer();

        mbeanServer.registerMBean(consumer, objectName);
} catch (MalformedObjectNameException e) { …
} catch (NotCompliantMBeanException e) { …
} catch (MBeanRegistrationException e) { …
} catch (InstanceAlreadyExistsException e) { ...
}
```

Detica

Demo 2a : Monitoring with JConsole

- JConsole -> MBeans tab
- MessageConsumer is underneath the com.company.application domain
- Attributes
  - running
  - messagesReceived and messagesProcessed

Detica

Demo 2b : Managing with JConsole

- Operations
  - start()
  - stop()
- Notice the thread name when logging occurs during method invocation
  - RMI TCP Connection

Detica

Exposing attributes revisited

- In addition to primitive types, method return values can be anything that is Serializable and available on the classpath of client
  - JConsole or a bespoke application

Detica

Demo 2c : Attributes with custom classes

- Expose the recentMessages attribute
  - returns Message[]
- Expose a statistics attribute
  - returns MessageStatistics
- Without extending JConsole's classpath, the value of the attributes is "Unavailable"
- How do you extend it?
  - jconsole -J-Djava.class.path=%JAVA_HOME%\lib\tools.jar;%JAVA_HOME%\lib\jconsole.jar;%YOUR_CLASSPATH%

Detica

Exposing attributes revisited

- Open MBeans provide some standard types
  - CompositeType : complex data structures
  - TabularType : arrays of CompositeType objects
- The real power of this is to ensure non-Java clients can attach to JMX and monitor applications
- MXBeans are like a cross between Standard and Open MBeans
  - Pattern based approach
  - Use of open types
  - Automatic conversion to open types

Detica

Demo 2d : Converting a Standard MBean to an MXBean

- This is useful because you get automatic conversion from bespoke return types to “open” types
- To convert our MessageConsumer
  - Rename the interface
- Side effects
  - Message[] becomes CompositeData[]
  - MessageStatistics becomes CompositeData

Detica

3. Enhanced monitoring with notifications

Detica

Monitoring a component

- How would you ensure an individual component is running?
  - Monitor stdout
  - tail –f <logfile>
  - Check messages are being consumed
  - Peek at it via JConsole (continually clicking “Refresh”) or via a bespoke application
- With JMX notifications, you can be pushed information rather than having to pull it

Detica

What are JMX notifications?

- Notifications can be broadcast from your component in response to something important/interesting/etc happening
  - Component started or stopped
  - Message received or processed
- Push rather than pull

Detica

Notifications

- Notifications are generic, having the following properties
  - A type
  - A timestamp
  - A sequence number
  - A message
  - Some user data (an Object)
  - A reference to the source of the notification

Detica

Notifications

- Notifications are instances of the javax.management.Notification (or a subclass)
- Constructors are available that let you specify the properties

Detica

Notification types

- A notification type is just a String
- Dot-separated notation can be used to inject user defined structure
- Convention : vendorname.component.event
- Example : company.messageconsumer.started

Detica

Broadcasting notifications

- To send notifications from an existing MBean, you need to implement
  - javax.management.NotificationEmitter
  - javax.management.NotificationBroadcaster
- Defines the ability to add and remove notification listeners
- Defines the ability to return information about the notifications emitted by the implementation

Detica

NotificationBroadcasterSupport

- Convenience implementation available
  - javax.management.NotificationBroadcasterSupport
- Provides support for managing notification listeners
  - Add/remove listeners
- Adds a sendNotification(Notification) method that calls back to listeners
  - Each listener's handleNotification() method is called

Detica

Demo 3 : Adding notification support

- Extend the NotificationBroadcasterSupport class
- Implement getNotificationInfo()
- Send notifications where necessary

Detica

Demo 3 : Listening with JConsole

- JConsole -> MBeans tab
- MessageConsumer is underneath the com.company.application domain
- Notifications
  - `company.messageconsumer.started`
  - `company.messageconsumer.stopped`
  - `company.messageconsumer.messagereceived`
- Subscribe and unsubscribe
  - Watch what happens when you start/stop the component

Detica

Considerations

- Notifications can be used for “real-time” status updates
  - There are no guarantees that they are actually real-time
- JMX notifications are not guaranteed
  - This depends on the underlying implementation
  - JMX notifications aren’t a replacement for messaging (e.g. JMS)

Detica

Considerations

- By default, all listeners are told to handle the notification in the same thread as the sender
  - i.e. our MessageConsumer message receiver thread
- If you want to send notifications and don't want to (potentially) lock up the thread while notifications are distributed, consider sending them in a separate thread
  - Java 6 makes this easy because you can specify an Executor when you instantiate a NotificationBroadcasterSupport instance, which is used for dispatching the callbacks
  - "DirectExecutor" used by default (i.e. in same thread)

Detica

4. Remote JMX

Detica

Remote JMX

- By default, the MBean server is only exposed to local clients
  - i.e. those connecting from the same host
- More often than not, it's useful to be able to connect to your JVMs remotely and manage them from a distance
- Multiple ways to do expose your MBean server for remote access
  - Automatically
  - Programmatically

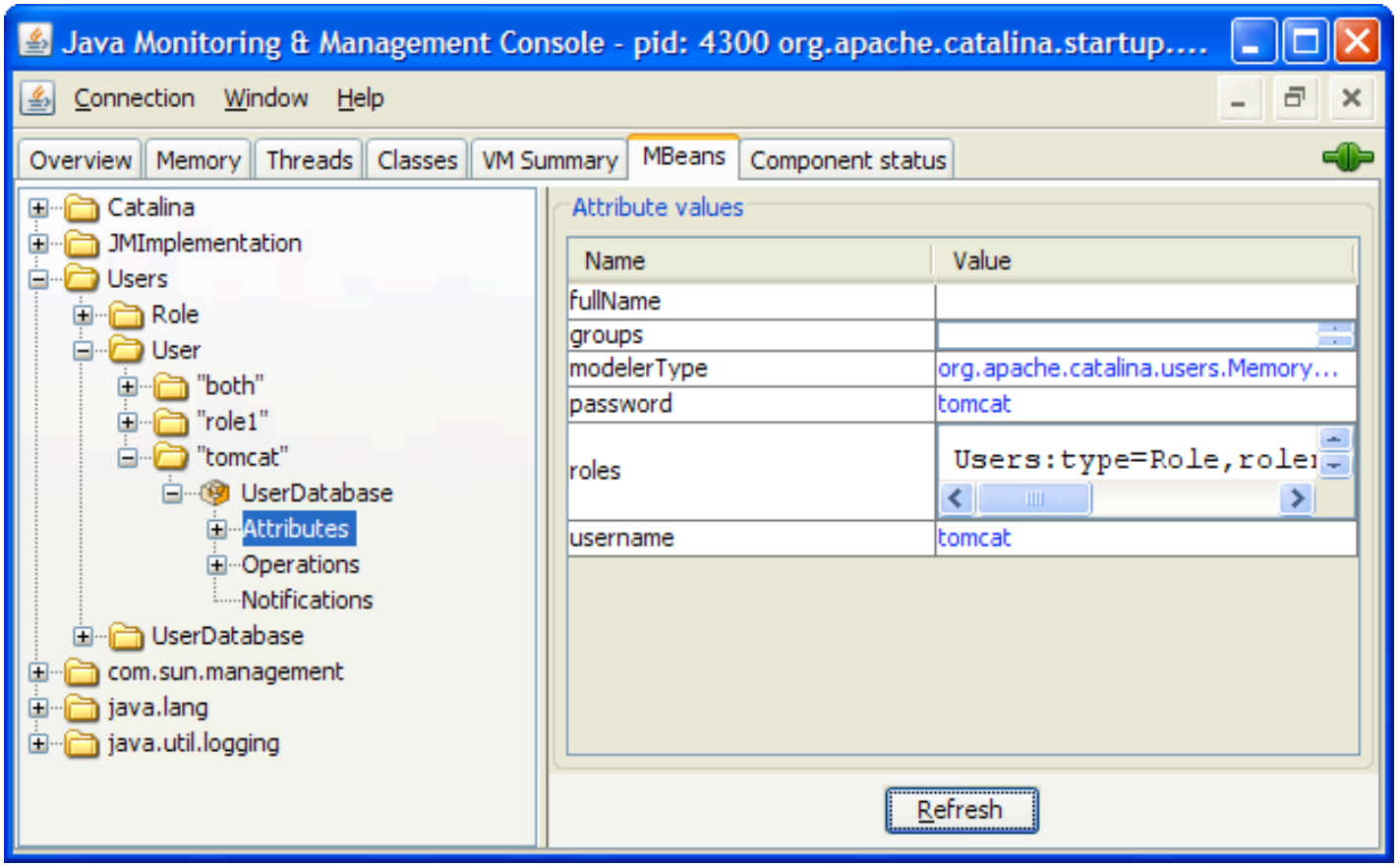
Detica

Demo 4a : Exposing an MBean server remotely

- An MBean server can be exposed remotely by specifying some parameters when starting up the JVM
- -Dcom.sun.management.jmxremote
- -Dcom.sun.management.jmxremote.port=9999
- -Dcom.sun.management.jmxremote.authenticate=false
- -Dcom.sun.management.jmxremote.ssl=false

Detica

If you don’t secure JMX



Detica

Demo 4b : Authentication and authorisation

- Addition JVM parameters can be used to turn on authentication and authorisation
- -Dcom.sun.management.jmxremote
- -Dcom.sun.management.jmxremote.port=9999
- -Dcom.sun.management.jmxremote.authenticate=true
- -Dcom.sun.management.jmxremote.ssl=false
- -Dcom.sun.management.jmxremote.password.file=etc\jmxremote.password
- -Dcom.sun.management.jmxremote.access.file=etc\jmxremote.access

Detica

Considerations

- Port clashing if you run a number of JMX enabled JVMs on a single host
- Security
  - Definitely require authentication to access your remote MBean server in production, particularly if it's not on a separate network
  - Consider securing the connection with SSL

Detica

Exposing an MBean server programmatically

- The programmatic way using JMX/RMI
- Bespoke URLs and ports
- See http://blogs.sun.com/lmalventosa/entry/mimicking_the_out_of_the for more details

Detica

Exposing an MBean server programmatically

- Bespoke authentication
- Plug in your own JMXAuthenticator instance
  - public Subject authenticate(Object credentials)

Detica

5. Use cases for enterprise applications

Detica

Component monitoring

- Component status (e.g. active, passive, running, etc)
- Attribute alerts
- Component heartbeats

Detica

Component management

- Starting and stopping components
  - Production management of your applications
  - A simple way to manage components during development and testing (e.g. when testing load balancing and failover)
- Throttling components
  - Telling components to slow down because they are saturating the resources

Detica

Log management

- How many times have you needed to switch from INFO to DEBUG and had to restart your JVM?
- Register a simple log manager component with the MBean server
  - Allows you to dynamically set log levels during application runtime

Detica

Configuration

- Have many times have you needed to change application configuration and had to restart your JVM?
- Register a configuration component with the MBean server and change configuration properties via JMX (e.g. JConsole)

Detica

Alerts and exceptions

- Use the JMX infrastructure to raise alerts when something happens
  - An attribute value passing a threshold
- Raise an alert when a threshold number of heartbeat notifications aren't received
- Broadcast exceptions via the JMX notification infrastructure
  - Only "really bad" exceptions

Detica

What do you expose via JMX?

- You can do a lot with the simple JMX concepts we've seen here so far, but where do you stop?
- Architectural components - yes
  - architectural components that you genuinely want to manage
- Domain objects, etc – no
  - There are more suitable mechanisms for accessing/exposing business data
  - Security issues, concurrency issues in updating the data, performance, scalability, etc
  - JMX is not a replacement for other remoting technologies

Detica

6. Programmatic access to MBeans

Detica

Programmatic access to MBeans

- MBeans can be located programmatically, using a bespoke Java application
- Local access
  - Create ObjectName and generate proxy
  - Query proxy
  - Subscribe to notifications
- Remote access
  - Create a remote connection to the MBean server
  - Create ObjectName and generate proxy
  - Query proxy
  - Subscribe to notifications

Detica

Generating a proxy to an MBean

- Proxy generation is very straightforward
  - Java 5 : MBeanServerInvocationHandler.newProxyInstance()
  - Java 6 : JMX.newMBeanProxy()
- Arguments are :
  - MBean server reference
  - ObjectName reference
  - Interface for the proxy that should be generated
  - True/false, depending on whether the MBean broadcasts notifications
- You can then call methods on the interface
  - MBean attributes or operations

Detica

Demo 6a : Accessing a local MBean

- Get a reference to the local MBean server
- Create an ObjectName to reference an MBean instance
- Generate a proxy
- Call methods on the proxy

Detica

Querying MBeans

- It is also possible to query for MBeans via an MBeanServer reference
  - All with a particular ObjectName
  - All MBeans in a particular domain
  - All MBeans of a particular type (i.e. querying on the type property)
- MBeanServer.queryMBeans() : Set<ObjectInstance>
- MBeanServer.queryNames() : Set<ObjectName>
- Arguments (for both) :
  - ObjectName
  - QueryExp (returned from factory methods on Query)

Detica

Demo 6b : Query the MBean server

- Multiple queries for ObjectNames
  - Specific ObjectName
  - Wildcard domain
  - Wildcard property list
  - Wildcard domain and property list

Detica

Subscribing to notifications

- Once you have a proxy to an MBean, you can subscribe to any notifications it broadcasts
  - Remember to specify true as the last parameter when generating proxies; this means the returned proxy implements NotificationEmitter
  - You need to perform the cast operation, unless your interface extends NotificationEmitter
- addNotificationListener()
  - NotificationListener
  - NotificationFilter
  - Object (for handback purposes)

Detica

Demo 6c : Subscribing to notifications

- Lookup the message consumer and subscribe to it's notifications
- Start up JConsole to compare notifications
- Stop/start the message consumer to see the notifications broadcast to all subscribers

Detica

Accessing a remote MBean

- All of what you've seen is possible with a remote MBean server
- Possible through the following classes
  - JMXServiceURL with a URL of service:jmx:rmi:///jndi/rmi://localhost:9999/jmxrmi
  - JMXConnector, created specifying the credentials of our secured MBean server

Detica

Demo 6d : Accessing a remote MBean

- As per the previous demo, but remote access

Detica

7. JMX and Spring

Detica

Expose any Spring bean via JMX

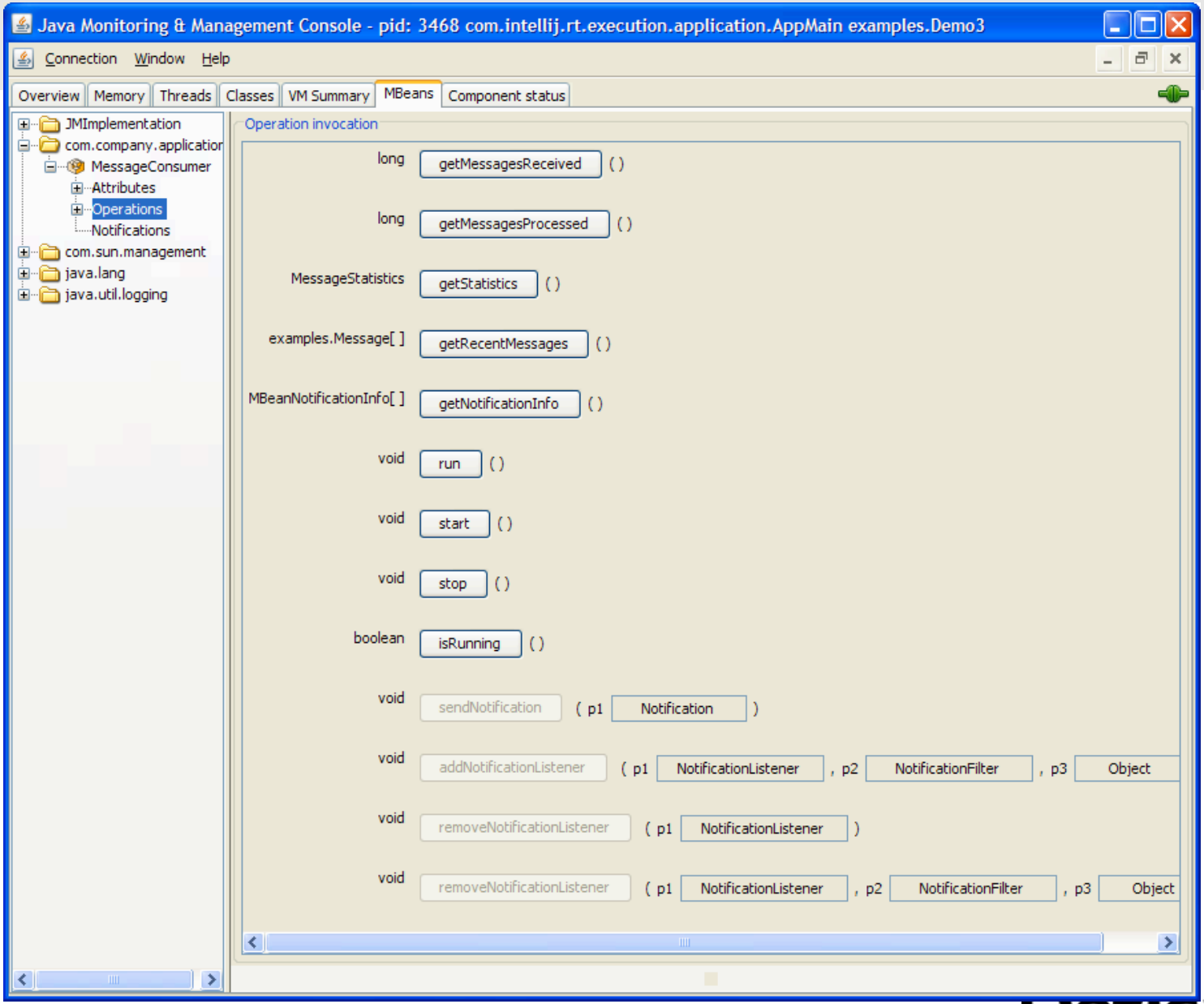
- Spring includes an MBeanExporter component
  - Allows you to expose *any* Spring bean via the JMX infrastructure
- You can forget…
  - Standard MBeans
  - looking up the platform MBean server
  - MBean registration

Detica

Demo 7a : MessageConsumer in Spring

- Look at the MBeanExporter class
- Create the MessageConsumer instance
- Export it into the Platform MBean server through Spring configuration

Detica


Java Monitoring & Management Console - pid: 3468 com.intellij.rt.execution.application.AppMain examples.Demo3
Connection
Window
Help
Overview
Memory
Threads
Classes
VM Summary
MBeans
Component status
JMImplementation
com.company.applicatior
MessageConsumer
Attributes
Operations
Notifications
com.sun.management
java.lang
java.util.logging
Operation invocation
long
getMessagesReceived
( )
long
getMessagesProcessed
( )
MessageStatistics
getStatistics
( )
examples.Message[ ]
getRecentMessages
( )
MBeanNotificationInfo[ ]
getNotificationInfo
( )
void
run
( )
void
start
( )
void
stop
( )
boolean
isRunning
( )
void
sendNotification
(p1
Notification
)
void
addNotificationListener
(p1
NotificationListener
, p2
NotificationFilter
, p3
Object
void
removeNotificationListener
(p1
NotificationListener
)
void
removeNotificationListener
(p1
NotificationListener
, p2
NotificationFilter
, p3
Object

Demo 7b : InterfaceBasedMBeanInfoAssembler

- Lets you explicitly specify which operations should be exposed via a Java interface
- Decouples you from the Standard MBean naming conventions
  - In a similar way to the StandardMBean class

Detica

Demo 7c : MetadataMBeanInfoAssembler

- use Commons Attributes or Java 5 annotations to control what gets exposed via metadata
  - @ManagedResource
  - @ManagedAttribute
  - @ManagedOperation

Detica

Demo 7d : MBean proxies

- Use Spring's MBeanProxyFactoryBean to generate proxies to MBeans
  - Local or remote

Detica

Demo 7e : Notifications, the Spring way

- Make your classes implement NotificationPublisherAware for publishing notifications easily
  - Spring IoC/DI
  - Allows easy unit testing with mock objects
- NotificationPublisherAware
  - public void setNotificationPublisher(NotificationPublisher)
- NotificationPublisher
  - public void sendNotification(Notification)
  - (same as NotificationBroadcasterSupport)

Detica

JMX and Spring

- You *could* forget everything you learnt about Standard MBeans and registering them
- I don’t think you should
- The Spring support is very good for getting you up and running quickly … but it has limitations

Detica

JMX and Spring

- The metadata associated with automatically exported MBeans isn't as rich as you would define yourself
- Automatically exporting an MBean without specifying an interface is useful but
  - You don't have control over what gets exposed
  - You need to use an interface if you intend to proxy the MBeans anyway

Detica

8. Extending JConsole

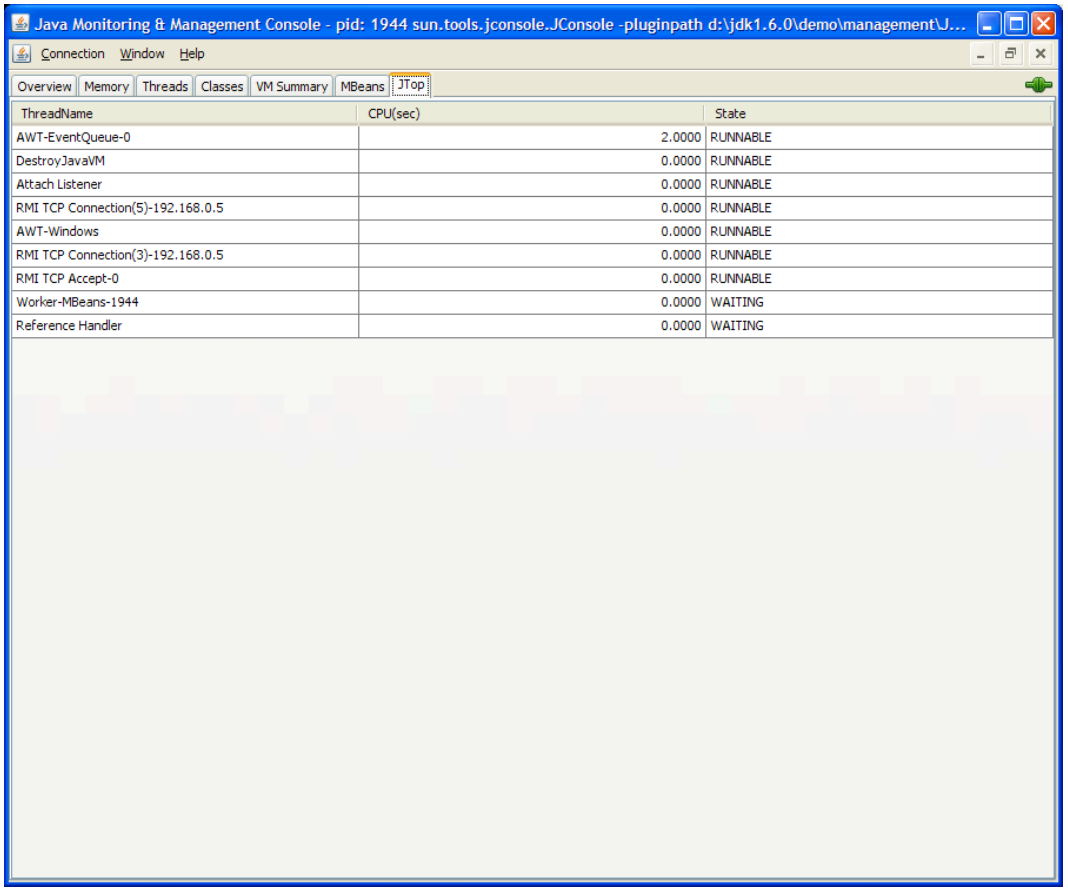
Detica

Java 6 introduces the JConsole API

- The JConsole API is a way to extend the functionality provided by the standard JConsole tool
- Lets you add more tabs to the main window
- Extend com.sun.tools.jconsole.JConsolePlugin
- JConsoleContext provides access to the underlying MBean server connection
- JConsole plugins are packaged up in a JAR file
  - META-INF/services/com.sun.tools.jconsole.JConsolePlugin file contains the fully qualified class name of the plugin

Demo 8a : JTop

- *NIX “top” for Java threads
- Included with the Java 6 SDK distribution



Detica

Demo 8b : A bespoke JConsole plugin

- A simple JConsole plugin to expose some information about the MessageConsumer component

Detica

9. Case study

Detica

Context

- Project for a major investment bank
- Team had chosen Java mainly because of resourcing constraints
- Essentially a “Greenfield” project
- Technology and architecture were to be decided

Detica

Technical architecture

- From an architecture perspective, the core part of the project was a collection of message handlers
  - JMS, but initially a proprietary messaging interface
- There were also some other components

Detica

Choosing a stack

- Java EE application server was overkill
- Lightweight (Java EE web tier) application server wasn't a good fit
- Lightweight standalone Java 5, SE applications configured with Spring was a good choice

Detica

Choosing a stack

- With a lightweight approach, we could get
  - Good testability
  - Database connection pooling
  - Transaction management
  - Horizontal scalability
  - etc
- But how would we manage the processes?

Detica

Monitoring and management

- The message handlers weren't message-driven (EJB) beans
- A bespoke container/component architecture was created
  - Components could be started and stopped at runtime
  - Each message handler instance was a component in its own right

Detica

Monitoring and management

- A top-level component interface defined the basic functionality
  - start(), stop() and restart()
- Each implementation could provide their own bespoke monitoring and management operations
  - Number of messages received
  - Component statistics

Detica

Monitoring and management

- Initially this was implemented through Spring JMX
  - MBeanExporter
  - Each component implemented a “ManageableComponent” interface
- This provided a very easy way to export functionality into the JMX infrastructure

Detica

Monitoring and management

- Soon became obvious that more was needed
  - More control over the remotability of the MBean server (port number clashes due to multiple JVMs on a single host)
  - More control over the object names (multiple components of the same type in the same JVM)
  - Ability to register and unregister components at runtime
    - This allowed us to (manually) scale the architecture to match the expected/actual load

Detica

Monitoring and management

- Back to basics approach
  - Standard MBeans
  - Manual lookup of MBean server
  - Bespoke registration of MBean instances
- All components were visible from JConsole, for monitoring and management
- Satisfied all our requirements … except one

Detica

Multiple JVMs

- Our system was horizontally scalable by simply starting up more JVMs
- How do you monitor and manage multiple JVMs?
  - Start up multiple JConsole sessions
  - Build a bespoke application

Detica

Sidenote : Cascading/federation

- Cascading/federation is planned for JMX 2.0
- Import MBeans from multiple MBean servers into a single MBean server
- See http://weblogs.java.net/blog/emcmanus/archive/2007/02/cascading_its_a.html for more details

Detica

Multiple JVMs

- We built a Java EE web application to provide basic monitoring and management across the multiple JVMs
  - List of active components
  - Component status
  - Links to manage the components (start, stop, restart)
  - Basic information (e.g. uptime, JVM memory, etc)

Detica

Multiple JVMs

- How do you know where the MBean servers are?
- Each JVM took control over exposing the platform MBean server
  - JMX/RMI, secured with a username/password with bespoke authentication
  - See http://blogs.sun.com/lmalventosa/entry/mimicking_the_out_of_the for more details

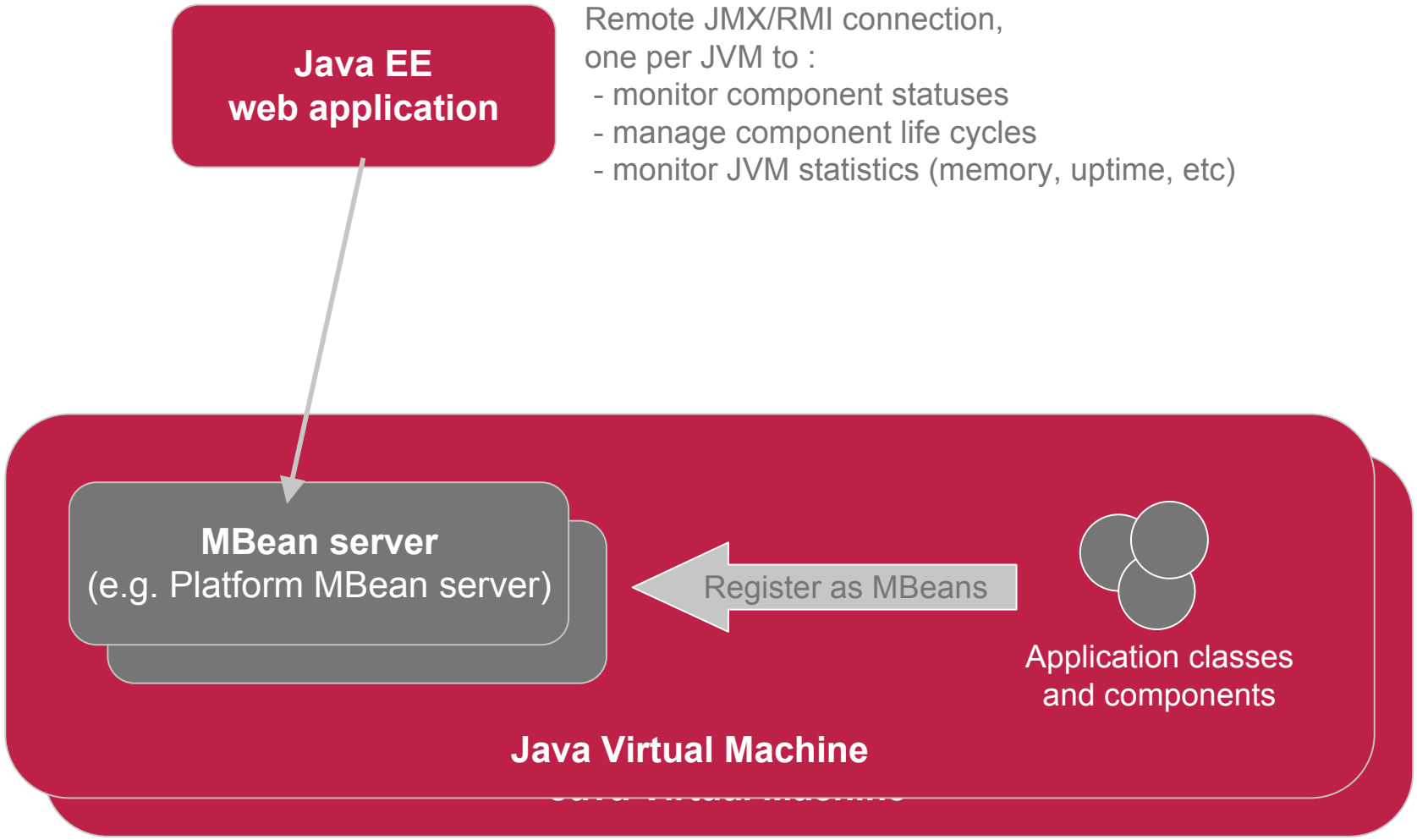
Detica

Multiple JVMs

- Each JVM registered the remote JMX/RMI URL in a database
- The webapp then connected to all registered MBean servers and queried the server to find all MBeans under a domain with a specific name/type
- Proxies to the MBeans using the basic “ManageableComponent” interface provided enough information for an composite view
  - isRunning(), start(), stop(), restart()

Detica

Multiple JVMs



Detica

10. Summary

Detica

Goals

- To give you the basics of JMX
  - Monitoring and managing a JVM with JConsole
  - Standard MBeans, notifications, programmatic access
- To show some of the use cases where JMX is applicable
  - Monitoring and management of enterprise components using different mechanisms, including Spring components
- To get you infected so that you leave here and add JMX support to your own applications
  - You've seen that JMX is simple and very powerful in enterprise application development

Detica

Links of interest

Detica

JMX in Java 6

- http://weblogs.java.net/blog/emcmanus/archive/2006/02/mustang_beta_an.html
- http://java.sun.com/javase/6/docs/technotes/guides/jmx/enhancements.html
- http://weblogs.java.net/blog/emcmanus/archive/2006/02/what_is_an_mxbe.html

Detica

Interesting JMX links

- What's the impact of using JMX? Negligible : http://weblogs.java.net/blog/emcmanus/archive/2006/07/how_much_does_i.html
- Java SE 6 Monitoring, Management, Diagnosability : http://weblogs.java.net/blog/mandychung/archive/2006/12/java_se_6_monit.html

Detica

Troubleshooting

- Error: Password file read access must be restricted
  - See http://blogs.sun.com/jmxetc/entry/the_hidden_command_that_would for a fix on Windows

Detica

Questions/discussion

Detica

Contact details

Simon Brown

Technical Architect, Global Financial Markets, Detica

http://www.simongbrown.com/blog/

simon.brown@detica.com

Head Office

Surrey Research Park

Guildford

Surrey

GU2 7YP

Tel: +44 (0)1483 816000

Fax: +44 (0)1483 816144

London Office

Arundel Great Court

2 Arundel Street

London

WC2R 3AZ

Tel: +44 (0)20 7812 4000

Fax: +44 (0)20 7812 4100

Cheltenham Office

1220 Lansdowne Court

Gloucester Business Park

Gloucester

GL3 4AB

UK

Tel: +44 (0)1452 632400

Fax: +44 (0)1452 632424

Detica