Creating Manageable Systems With JMX, Spring, AOP, and Groovy

Vladimir Vivien

Sr. Software Engineer
Simplius, LLC
http://simpli.us/

TS-1106

Goal

Build runtime manageable systems

Explore how to build manageable systems using Java™ Management Extensions (JMX™), AOP, and Groovy within the context of a Spring container.

Agenda

- Motivation
- Introduction to JMX Technology
- JMX Technology and the Spring Framework (v.2)
- JMX Technology With Spring AOP
- JMX Technology Notification and Spring
- Agile JMX Technology Notification Handling
- Put It All Together: Food Planet Demo
- Summary
- Q&A

Agenda

- **Motivation**
- Introduction to JMX Technology
- JMX Technology and the Spring Framework (v.2)
- JMX Technology With Spring AOP
- JMX Technology Notification and Spring
- Agile JMX Technology Notification Handling
- Put It All Together: Food Planet Demo
- Summary
- Q&A

Perilous Driving

- Would you drive this car?
  - No speedometer
  - No gas gauge
  - No engine temp gauge
  - No instrument panel
  - No signal flashers
- Most deployed apps have similar shortcomings
  - System is a black box
  - No runtime visibility
  - No way to see app's states

Motivation

What do we want to achieve

- Better visibility of system at runtime
- Go beyond simple event log files
- Ability to monitor
  - Expose states/health of application in real time
  - Take preventive measures where possible
  - React intuitively and quickly to changes and requirements
- Greater management
  - Interactively control application in user-friendly way
  - Ability to change operational parameters on the fly
  - Avoid down time for critical applications

Agenda

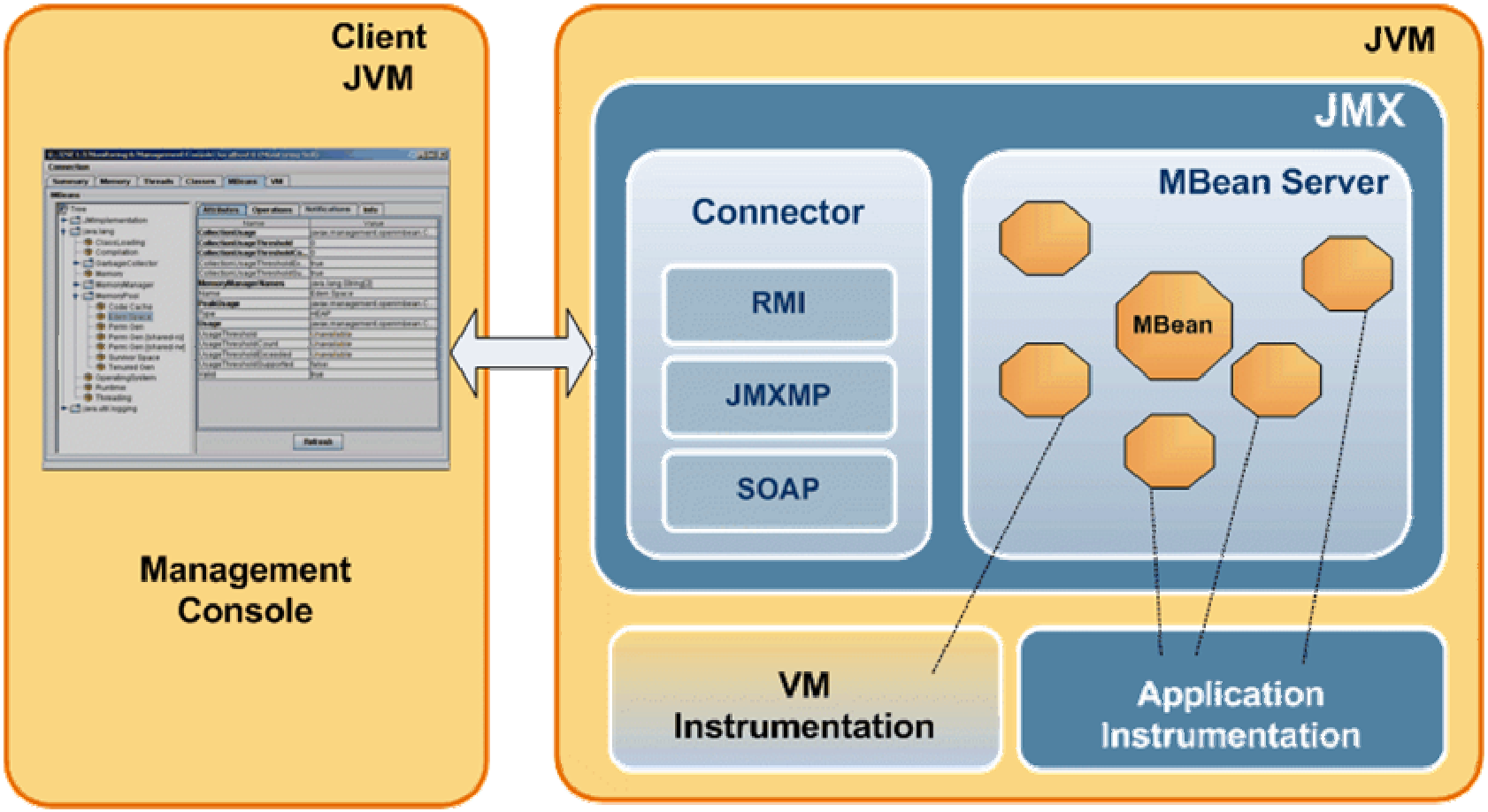
- Motivation
- **Introduction to JMX Technology**
- JMX Technology and the Spring Framework (v.2)
- JMX Technology With Spring AOP
- JMX Technology Notification and Spring
- Agile JMX Technology Notification Handling
- Put It All Together: Food Planet Demo
- Summary
- Q&A

JMX Technology

What is JMX Technology?

- Java Management eXtension
- Mature API [originally Java Specification Request (JSR)-3]
  - Standard Java API for exposing management
  - Includes a server and wire protocols for connection
  - Provides several monitoring and a timer components
- Main usage is management and monitoring
- Part of core Java platform starting with version 5
  - VM information exposed through JMX technology
  - Instrument and expose your own application
  - Management standard for Java Platform, Enterprise Edition (Java EE platform) artifacts (JSR 77)

JMX Management Architecture



JVM™ = Java Virtual Machine
The terms “Java Virtual Machine” and “JVM” mean a Virtual Machine for the Java™ platform.

The MBean Server

The management context

- Container for management components
  - Allows registry, discovery, and query of components
  - Handles interaction with components from client
  - Exposes component operations and properties
- Provides event bus
  - Register to receive event notification
  - Broadcast events to registered components
- Server is exposed as management component

The Management Bean (MBean)

How management information is exposed

- Exposes managed resources in standard way
- MBean API:
  - *Standard MBean* interface
    - The simplest way to implement management
    - Requires naming pattern (i.e. xxxMBean)
  - Others include *Dynamic*, *Model*, and *Open MBean*
- Registered in the MBean Server
  - Makes it visible to management client
  - Exposes *attributes* (JavaBeans™ architecture-style getters/setters)
  - Expose *operations* (interface methods)
  - Event *Notifications*

The Object Name

- `javax.management.ObjectName`
- Uniquely identifies registered MBeans
- Supports flexible naming strategy
- Format
  - `[domain]:key=value[,key=value]*`
- Example
  - *demo.service:type=Greeting,description=greets the world*
- Use by management tool extensively

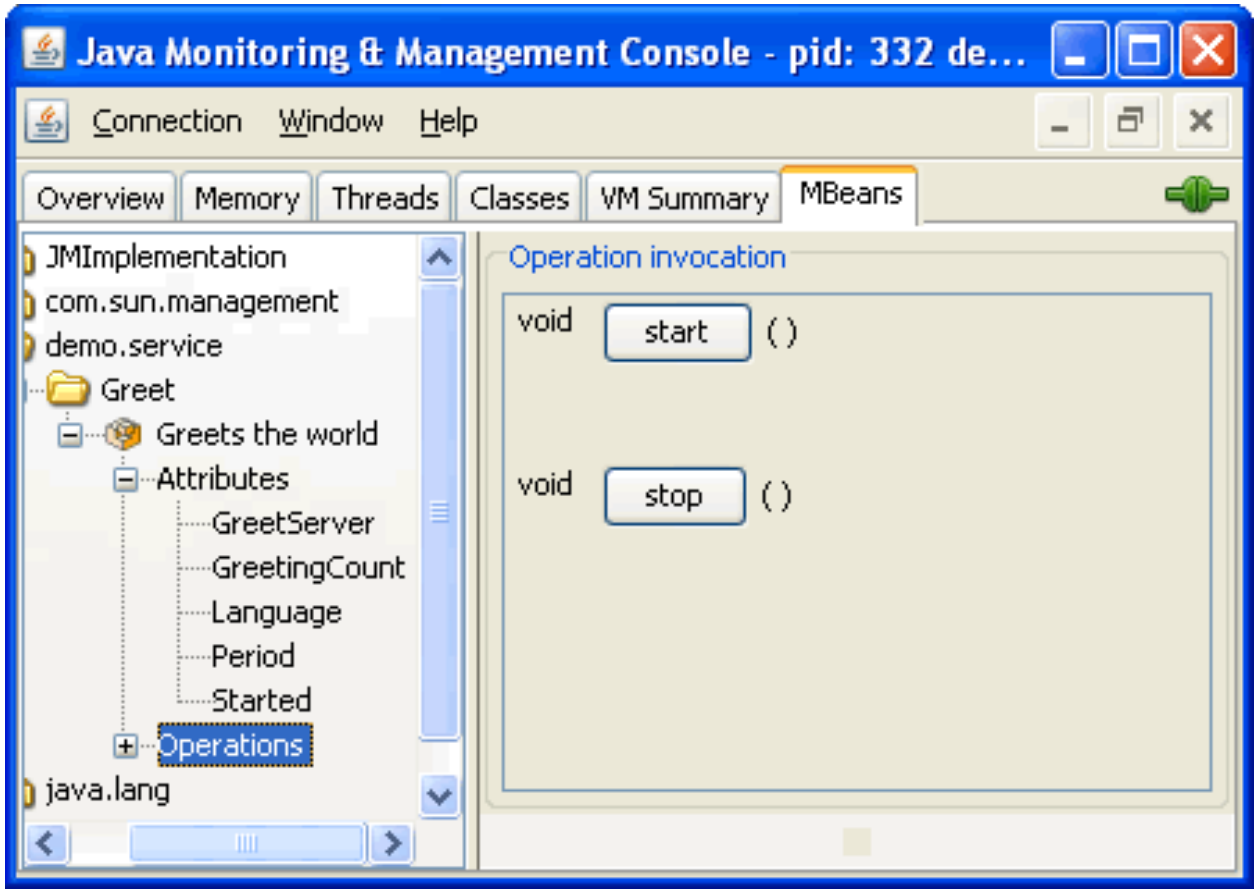
Exposing an MBean—Code

1
```
public interface GreetServiceMBean {
  public void setLanguage(String lang);
  public void start();
  public void stop(int val);
}
```

2
```
public class GreetService implements GreetServiceMBean {
  public void setLanguage(String lang){...}
  public boolean start() {...}
  public void stop(int val) {...}
}
```

3
```
public void RegisterMBean () throws Exception {
  MbeanServer svr = ManagementFactory.getPlatformServer();
  ObjectName objName =
       new ObjectName("demo.service:type=Greet...");
  svr.registerMBean(new GreetService(), objName);
}
```

Exposing an MBean—Console View

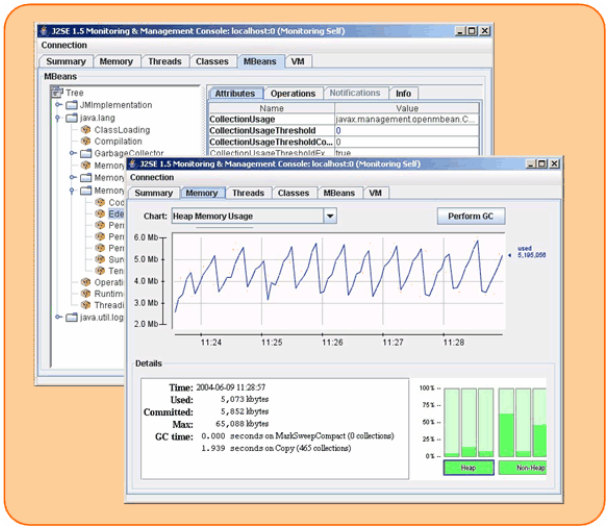


Managing the VM

- Starting with Java platform v.5, VM includes instrumentation
- VM exposes numerous information as MBeans
  - Class loading information
  - Memory consumption
  - Garbage collection
  - Threads
  - Etc.
- JConsole makes a formidable profiling tool
- Ability to look into VM's activities with no code

Management Console—JConsole

- Interactive desktop management tool
- Introduced in Java Development Kit (JDK™) 5 release
- Profiles memory, threads, GC
- Supports for remote connection
- Attaches to running VM
  - Java platform v.5 requires switch
  - `-Dcom.sun.management.jmxremote`



DEMO

Exposing MBeans/JConsole

Agenda

- Motivation
- Introduction to JMX Technology
- **JMX Technology and the Spring Framework (v.2)**
- JMX Technology with Spring AOP
- JMX Technology Notification and Spring
- Agile JMX Technology Notification Handling
- Put It All Together: Food Planet Demo
- Summary
- Q&A

Why Spring?

- Extensive support for JMX technology
- Lightweight and portable
- Simplified development model using POJO's
- Platform for modular development
- Consistent Infrastructure API's
  - Enterprise Java technology—Enterprise JavaBeans™ (EJB™) architecture, Java Message Service (JMS) API, J2EE Connector Architecture (JCA), Web Service, JMX technology
  - Data Access—Java DataBase Connectivity (JDBC™) software, Hibernate, TopLink, Java Data Objects (JDO), Java Persistence API (JPA)
  - Web—Spring MVC, Tapestry, JavaServer™ Faces, Struts, etc.
  - Others—AOP, scripting, scheduling, concurrency
- http://springframework.org/

JMX Technology and Spring

- Declaratively registers any POJO as MBeans
- No special interface or naming patterns required
- Spring provides the MBean Exporter component
- Full control and flexibility:
  - Support several strategies for ObjectName
  - MBeanInfoAssembler API
    - Supports different strategies for exporting MBeans
    - Use ***list of interfaces*** for management
    - *Use* ***Method names*** to export for management
    - Use ***code-level annotation*** *to customize export*
  - *Specify MBean registration behavior*

JMX Technology and Spring (Cont.)

- *Wire POJO's as JMX technology notification listeners*
- *Use Spring's supplied MBean server or provide your own (defaults to VM's)*
- *Auto detects classes with JMX technology naming pattern*
- *MBeans can take advantage of DI container*
  - *Declarative injection of MBean dependencies*
  - *Easily establish relationship between MBeans*

Simple Spring JMX Technology Exporter

1

```
public class GreetService {
  public void setLanguage(String lang){...}
  public boolean start() {...}
  public void stop(int val) {...}
}
```

spring-context.xml

```
<bean id="greetSvc" class="demo.GreetService"/>
<bean id="exporter"
  class="org.springframework.jmx.export.MBeanExporter">

  <property name="beans">
    <map>
      <entry key="demo.service:type=Greet ..."
        value-ref="greetSvc"/>
    </map>
  </property>

</bean>
```

2

Spring Exporter

ObjectName

Metadata Info Assembler

Using annotation to create management interfaces

- MetaDataMBeanInfoAssembler class
- Builds MBeans using code-level annotations
  - Supports ***Java platform annotations***
  - Supports ***Commons Attributes*** annotations
  - Automatically registers beans as MBeans that have the @ManagedResource annotation
- Granular control of MBean attributes/operations
  - Use @ManagedAttribute annotation for bean methods
- However, introduces dependency on Spring

Java Platform v.5 Annotation Example

1

```
@ManagedResource(objectName="demo.service:type=Greet ...")
public class GreetService {
  @ManagedAttribute
  public void setLanguage(String lang){...}    ...
}
```

2

```
<bean id="greetSvc" class="demo.GreetService"/>
<bean
  id="attribSrc"
  class="...export.annotation.AnnotationJmxAttributeSource"/>

<bean id="exporter" class="...export.MBeanExporter">
  <property name="autodetect" value="true"/>
  <property name=”assembler”>
    <bean class="...export.assembler.MetadataMBeanInfoAssembler">
       <property name="attributeSource" ref="attribSrc"/>
    </bean>
  </property>
</bean>
```

DEMO

JMX Technology and Spring

Agenda

- Motivation
- Introduction to JMX Technology
- JMX Technology and the Spring Framework (v.2)
- **JMX Technology With Spring AOP**
- JMX Technology Notification and Spring
- Agile JMX Technology Notification Handling
- Put It All Together: Food Planet Demo
- Summary
- Q&A

JMX Technology and AOP

Leveraging AOP for management

- Aspects can provide clean separation of concerns
- Aspects make infrastructural concerns, such as management, transparent to other concerns
- Business logic is cleaner
  - Remove dependency on management code
  - Reduce/eliminate code entanglement
- AOP suited for JMX technology monitoring
  - Aspects collects system states as application executes
  - Aspects channel instrumented data to **MBeans**

Spring AOP

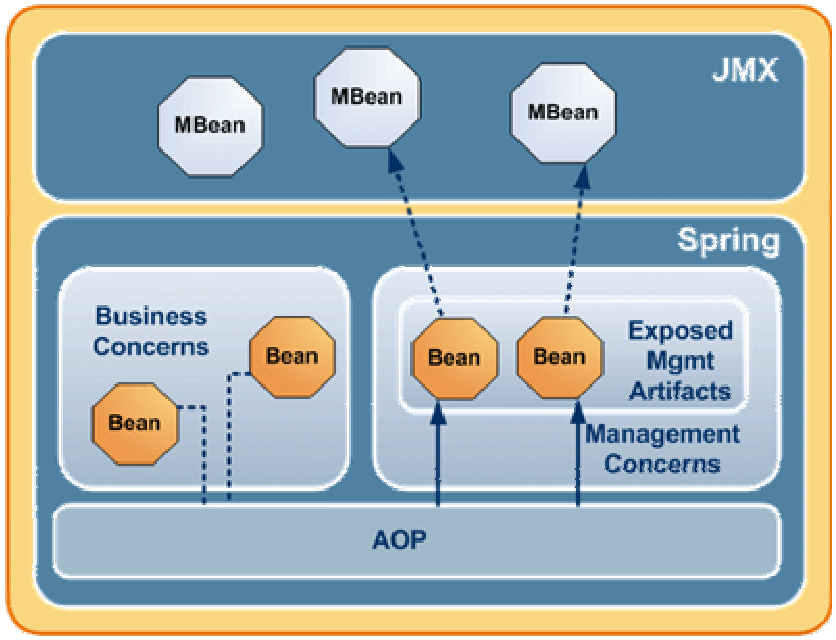
The easier way to aspects

- A proxy-based implementation
  - Only supports method invocation interception
  - Intercept methods calls on Spring beans
  - Spring uses the popular AspectJ language
  - Spring aspects are applied at runtime
- Spring makes it easy to create aspects
  - Declaratively—any Spring bean may be an aspect
  - Annotation—supports @AspectJ code-level annotation
  - Annotated class can be used with AspectJ
  - http://springframework.org/documentation/—Chapter 6

JMX Technology and Spring AOP

Leveraging Spring AOP for management

- Augment management infrastructure
- Transparently aggregate data for instrumentation
- Push instrumented data to management beans



Using Spring AOP

1

```
public class GreetingAspect {
  private GreetingMbean mbean;
  public void beforeGreeting() {...}
  public void afterGreeting() {...}
}
```

Management Bean

Aspect bean

spring-context.xml

Intercepts

2

```
<bean id="aspectBean"
  class="demo.GreetAspect"/>
<aop:config>
  <aop:pointcut id="myPointCut"
     expression="execution(* demo.GreetServer.greet(..)) "/>
  <aop:aspect id="aspect" ref="aspectBean">
    <aop:before
      pointcut-ref="myPointCut"
      method="beforeGreeting"/>
    <aop:after
      pointcut-ref="myPointCut"
      method="afterGreeting"/>
  </aop:aspect>
</aop:config>
```

Agenda

- Motivation
- Introduction to JMX Technology
- JMX Technology and the Spring Framework (v.2)
- JMX Technology With Spring AOP
- **JMX Technology Notification and Spring**
- Agile JMX Technology Notification Handling
- Put It All Together: Food Planet Demo
- Summary
- Q&A

JMX Technology Notification Model

- JMX technology has a rich notification model
- Integral part of JMX technology
- Interfaces included in JMX technology notification model
  - Notificationcontent of notification
  - NotificationBroadcaster – source of event notification
  - NotificationListener – recipient of notification
  - NotificationFilter – allows filtering of notifications
- The MBean server broadcasts numerous events
- Your Mbeans can broadcast/listen for events
- Management tools can subscribe to notifications

Notification Object

Looking inside the notification Object

- Interface used for event notification
- Event broadcasters pass an instance of this class to event listeners
- Contains
  - String representing notification type
  - A JMX technology ObjectName reference to broadcaster
  - Sequence number for the event
  - Time stamp when notification was created
  - A string message about the event
  - A broadcaster supplied data object

JMX Technology Listener Registration

1
```
public class MyTimer extends javax.management.timer.Timer{}
```

2
```
public class TimerLstnr implements NotificationListener{
  public handleNotification(Notification n, Object obj){...}
}
```

3
```
public class RegisterListener throws Exception{
  demo.MyTimer timer = new demo.MyTimer();
  Notification n = new Notification("timer.heartbeat",
      null, new Date(System.currentMillis()));
  timer.setNotification(n);
  timer.setPeriod(3000);

  NotifycationFilterSupport f = new NotificationFilterS...
  f.enableType("timer.heartbeat");
  TimerLstnr lstnr = new TimerLstnr();
  timer.addListener(lstnr,f,null);
}
```

Listener Registration With Spring

```
<bean id="myTimer" class="demo.MyTimer">
  <property name”period” value=”3000”/>
</bean>

<bean id="exporter" class="...export.MBeanExporter">
  <property name="beans">
  <map><entry key="jmx.timer:type=Timer" value-ref="myTimer"/></map>
  </property>

  <property name="notificationListenerMappings">
    <map>
      <entry key="jmx.timer:type=Timer">
          <bean class="demo.TimerLstnr"/>
      </entry>
    </map>
  </property>
</bean>
```

Agenda

- Motivation
- Introduction to JMX Technology
- JMX Technology and the Spring Framework (v.2)
- JMX Technology With Spring AOP
- JMX Technology Notification and Spring
- **Agile JMX Technology Notification Handling**
- Put It All Together: Food Planet Demo
- Summary
- Q&A

Agile Notification Handling

Motivation for using dynamic language for management

- Mechanism to react to JMX technology notifications
- Agile adaptation to changes in business conditions and requirements
- Autonomous reactions
  - Capture business rules and work flows
  - Apply rules automatically with no intervention
- Extends system functionalities while core infrastructure remains intact
- Ability to react preventively to adverse conditions

Why Groovy?

Using Groovy as glue code for monitored events

- *Integrates well with the Spring Framework*
- Popular language and described as
  - "***Agile dynamic language*** *for the Java platform* ***inspired*** *by* ***Python***, ***Ruby***, *and* ***Smalltalk***..."
- *Compiles directly into Java bytecode*
  - *Groovy scripts import Java objects (vice versa)*
- *Some interesting language features include:*
  - *Closures*
  - *Duck typing*
  - *optional line terminator and method parentheses*
  - *Extended syntactical support for collections*

Groovy and Spring

Integrating Groovy and Spring

- Spring 2.0 introduces support for Groovy
  - Also supports *JRuby* and *BeanShell*
- Declaratively wire Groovy bean into Spring
- Groovy bean can use getter/setter injection
- Groovy bean must implement Java platform interface
  - Spring requires an interface for proper typing
- Refreshable flag updates Groovy bean
- http://springframework.org/documentation—Chapter 24
- http://groovy.codehaus.org/

Groovy Bean With Spring

1

```
public interface Action {
  public void act();
  public setValue(String
v);
}
```

Java Interface

```
class GAction extends Action {
  public act() { ... }
  String value
}
```

Groovy Script

spring-context.xml

2

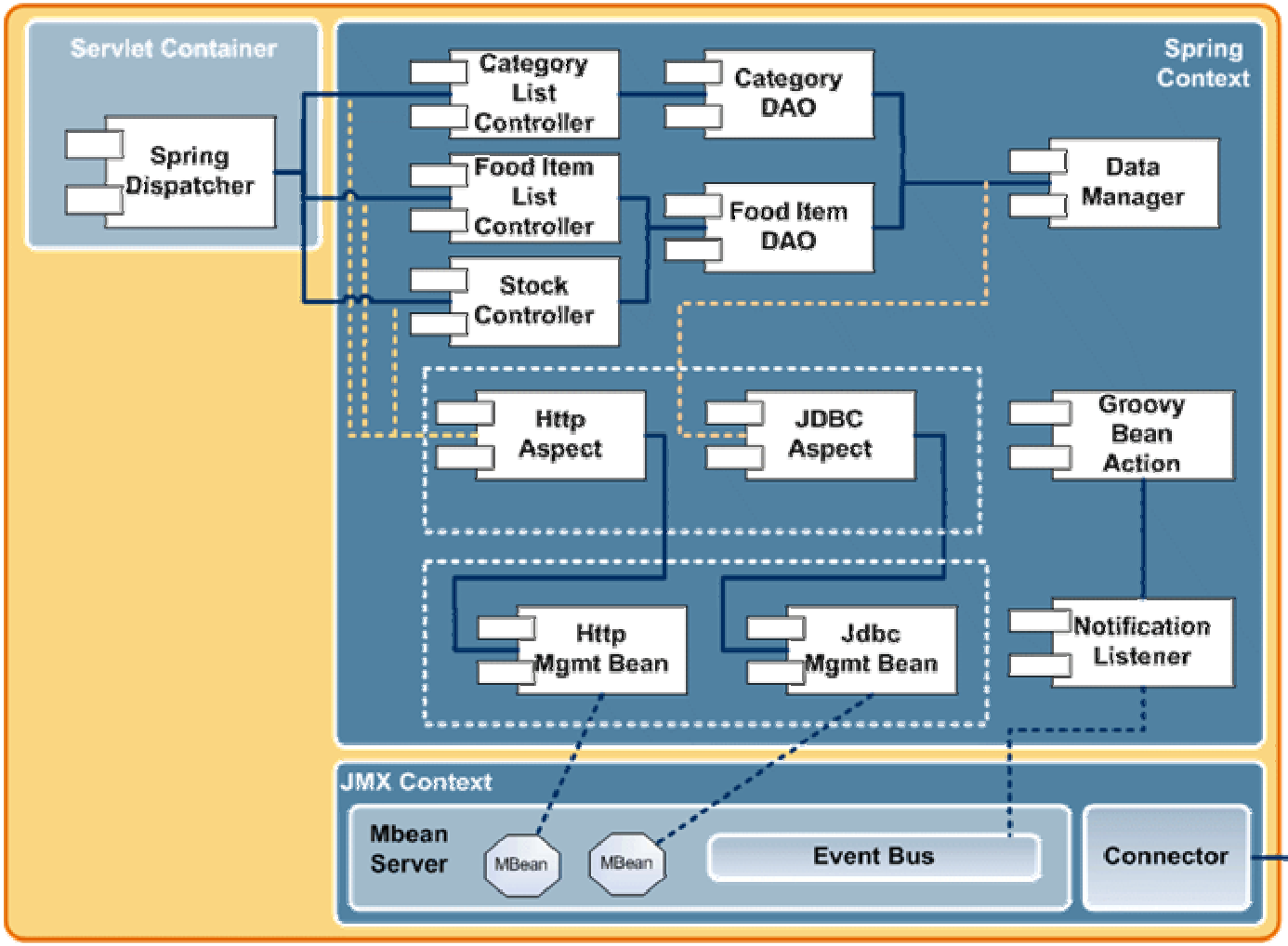
```
<beans>

  <lang:groovy id="action" refresh-check-delay="5000"
    script-source="/META-INF/script/GAction.groovy">
    <lang:property name="value" value="RUN"/>
  </lang:groovy>

  <bean id="listener" class="demo.GreetingListener">
    <property name="action"><ref bean="action"/></property>
  </bean>

</beans>
```

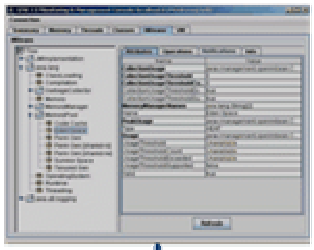
Groovy Bean

Agenda

- Motivation
- Introduction to JMX Technology
- JMX Technology and the Spring Framework (v.2)
- JMX Technology With Spring AOP
- JMX Technology Notification and Spring
- Agile JMX Technology Notification Handling
- **Put It All Together: Food Planet Demo**
- Summary
- Q&A

Food Planet Components





Food Planet Application

Using JMX technology, Spring AOP, and Groovy together

- Application runs within a Spring context
  - JdbcTemplates provides database access
  - Uses Spring MVC Controllers for web access
  - Http Access Management
    - Aspect intercepts requests to MVC Controllers
    - Aspect aggregates data for instrumentation
    - Http MBean provides monitoring/control of Http access
  - Inventory Database Management
    - Aspect intercepts database during purchase
    - Aspect aggregates inventory data for instrumentation
    - Inventory MBean monitors inventory level
    - When item less then threshold sends notification
    - Groovy bean is used to manage inventory level events

DEMO

Food Planet Online Store

JMX Technology Design Issues

Designing for management

- Management beans should be lightweight
- Develop meaningful MBean naming strategy
  - Management consoles use bean name extensively
  - Use JMX technology domain to group similar beans
  - Use name's key/value attribute to categorize beans
- Usability issues
  - Expose editable values (numbers, string, etc.)
  - Avoid null attributes
  - Avoid MBean methods with large parameter count

JMX Technology Design Issues (Cont.)

- Create consistent management model
- Monitoring
  - Use AOP to collect instrumented values
  - Use JMX technology event bus to handle monitored events
- Application Control
  - Provide life cycle and functional controls
  - Use action verbs to describe push-button functions
- Configuration
  - Update application parameters at runtime
  - Provide feedback attributes to reflect changes

Agenda

- Motivation
- Introduction to JMX Technology
- JMX Technology and the Spring Framework (v.2)
- JMX Technology With Spring AOP
- JMX Technology Notification and Spring
- Agile JMX Technology Notification Handling
- Put It All Together: Food Planet Demo
- **Summary**
- Q&A

Summary

- Log files are no longer enough for the critical applications
- Ability to monitor system states and health in real time
- React preventively to adverse conditions
- JMX technology provides a robust management API to meet stringent management needs
- JMX technology includes a management server, event notification mechanism, monitoring service, and connectors for management clients
- It is available in JDK software starting with Java platform v.5
- JMX technology and JConsole management console provide a full-featured diagnostic, monitoring, and VM profiling tool

Summary (Cont.)

- JMX technology integrates well with the Spring Framework
- Any Spring POJO can be exposed for management
- Spring AOP can augment management infrastructure of system by transparently aggregate instrumented data
- The JMX technology Notification offers an impressive set of features use to broadcasts events to the JMX technology runtime
- Combined with a dynamic language such as Groovy, JMX technology can create agile mechanism to react to monitored event notifications in more expressive and rich manner

For More Information

- JMX Technology
  http://java.sun.com/products/JavaManagement/
- Spring
  http://www.springframework.org/documentation
- Groovy
  http://groovy.codehaus.org/
- AOP
  http://eclipse.org/aspectj

For More Information

- Tools
  - Jconsole
    http://java.sun.com/javase/6/docs/technotes/tools/share/jconsole.html
  - MC4J
    http://mc4j.org/
  - Glassbox
    http://glassbox.com/
- Broadway Project
  - https://broadway.dev.java.net/

Q&A

Creating Manageable Systems With JMX, Spring, AOP, and Groovy

Vladimir Vivien

Sr. Software Engineer
Simplius, LLC
http://simpli.us/

TS-1106