Spring Framework Appendices
Table of Contents

2

Notes on Using HSQL

- *HSQL DB* is an open source Java-based relational database used for this course
- It is easy to setup and work with
  - Consists of one Jar: ***hsqldb.jar***
- To start the DB:
  - From within Eclipse
    - Expand the HSQLDB project and double-click ***StartDB.bat***
- From a command-prompt:
  - From within the <workspace>/HSQLDB/data directory, type:

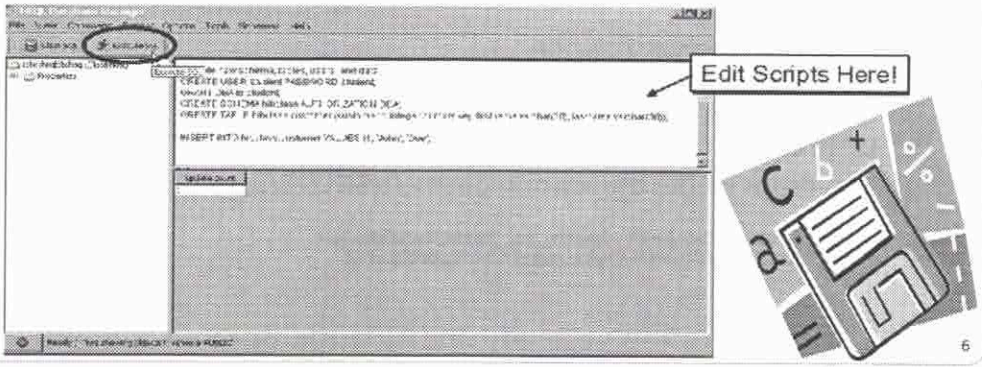
    java -classpath <path_to_jar>/hsqldb.jar org.hsqldb.Server

4

Notes:

Notes on Using HSQL ...

- To run a script or query, start the GUI Database Mgr
  - Connect with type 'HSQL Database Engine Server'
  - Select **File → Open Script...** → (locate .script file)
  - Press **Execute SQL** button on the Toolbar



Notes:

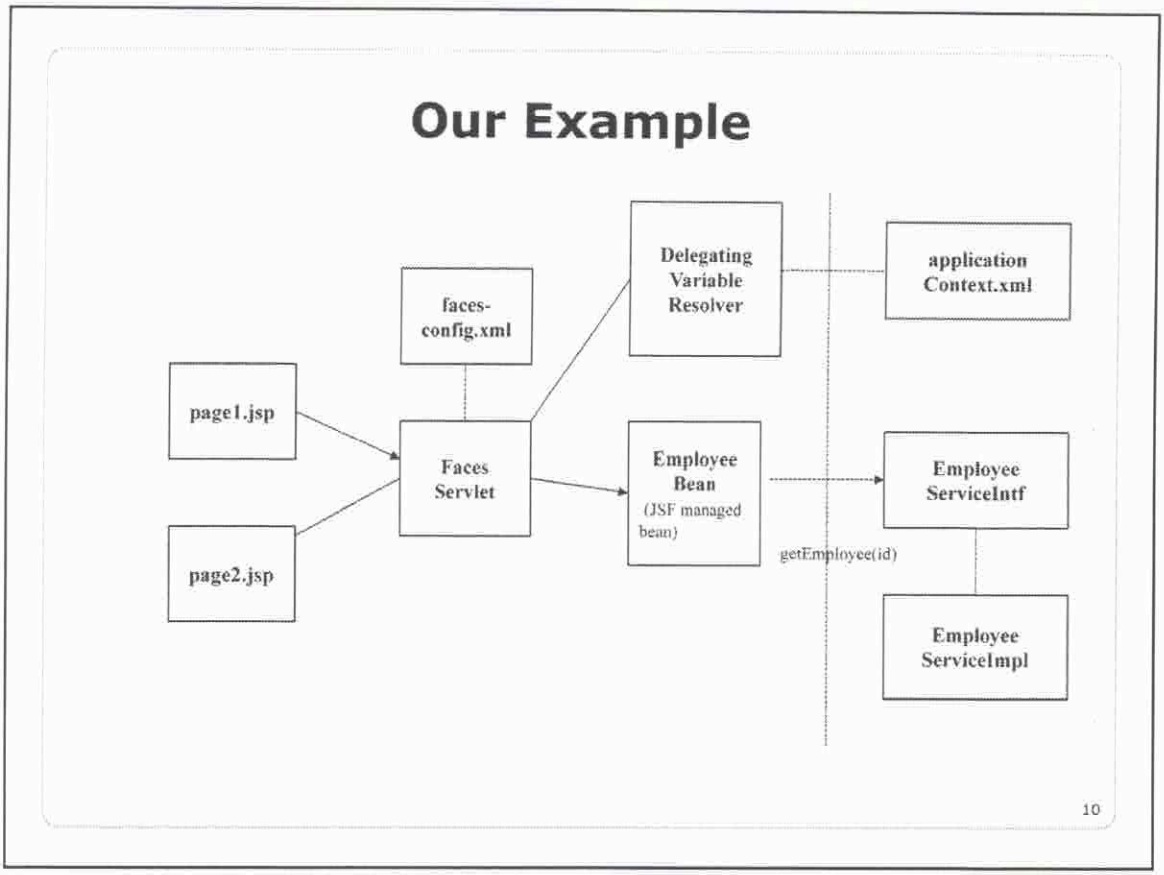
Scripts may also be manually typed or even copied and pasted into the script editor and then executed.

Web Tier Integration and JSF Overview

- Spring-Enabling JSF-based Apps
- Spring to JSF Integration
- FacesContextUtils
- JSF to Spring Integration

8

Notes:


Our Example
faces-config.xml
Delegating Variable Resolver
application Context.xml
page1.jsp
Faces Servlet
Employee Bean
(JSF managed bean)
Employee ServiceIntf
page2.jsp
getEmployee(id)
Employee ServiceImpl
10

Notes:

The Managed Bean: EmployeeBean

```
public class EmployeeBean {

    private EmployeeServiceIntf employeeService;
    private int id;
    private Employee employee;

    // getters and setters for these properties

    public String find() {
        employee = employeeService.getEmployee(id);
        return "success";
    }
}
```

This is a Spring bean injected by the JSF container

12

Notes:

Notice the reference in the managed bean to the Spring-based EmployeeService. This will be initialized when Spring and JSF start up.

applicationContext.xml

```
<beans ...>
    <bean id="dataSource"
          class="org.apache.commons.dbcp.BasicDataSource"
          destroy-method="close" >
          <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>
                    <property name="url" value="jdbc:hsqldb:hsql://localhost"/>
                    <property name="username" value="student"/>
                    <property name="password" value="student"/>
    </bean>
    <bean id="employeeService"
          class="com.company.springclass.services.EmployeeServiceImpl">
          <property name="dataSource">
                    <ref local="dataSource"/>
          </property>
    </bean>
</beans>
```

14

Notes:

The Spring config file defines a bean called employeeService. This looks the same as it did in previous sections.

FacesContextUtils

- Another way to access Spring beans in JSF apps is to use *FacesContextUtils*
- A Spring-based **JSF Helper Utility**

```
EmployeeServiceIntf employeeService = (EmployeeServiceIntf)
        FacesContextUtils.getRequiredWebApplicationContext(
        FacesContext.getCurrentInstance()).
        getBean("employeeService");
```

16

Notes:

Managing JSF Beans in Spring

- The Approach:
  1. Set up a Spring *RequestContextListener*
  2. Set up the *DelegatingVariableResolver* in *faces-config.xml*
  3. Move the managed bean definition into the Spring config file
  4. Remove the managed bean from *facesconfig.xml*

18

Notes:

faces-config.xml

The ***employeeBean*** definition has been removed from here

```
<faces-config>
    <application>
    <variable-resolver>
            org.springframework.web.jsf.DelegatingVariableResolver
    </variable-resolver>
   </application>

<navigation-ruls>
    <from-view-id>/page1.jsp</from-view-id>
    <navigation-case>
            <from-outcome>success</from-outcome>
            <to-view-id>/page2.jsp</to-view-id>
    </navigation-rule>
</navigation-rule>

</faces-config>
```

20

Notes:

Summary

- Because both JSF and Spring have injection capabilities, there is an overlap of responsibilities
- There are 3 ways to integrate Spring and JSF applications:
  - Use a DelegatingVariableResolver for Spring to JSF integration
  - Use the FacesContextUtils class provided by Spring for Spring to JSF integration
  - Use a RequestContextListener for JSF to Spring integration

22

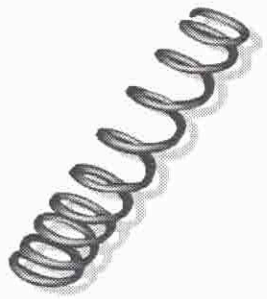
Notes:

This page intentionally left blank!

24

Spring Remoting Overview

- Spring Remoting with RMI
- Remoting Configuration
- Remoting Example



26

Notes:

Spring Remoting Services

- Spring supports distributed objects in several different ways:
  - RMI
  - JAX-RPC
  - JAX-WS
  - HTTP-Invoker

28

Notes:

There are several ways of remoting within Spring. While web services are commonly encountered, and Spring supports them with its JAX-WS SOAP capabilities, we will only tackle one form of remoting here: RMI.

Exposing the DayOfWeek Service

```
public interface DayOfWeek {

    public String getDayOfWeek(int year, int month, int day);
}
```

```
public class DayOfWeekImpl implements DayOfWeek {

    public String getDayOfWeek(int year, int month, int day) {

        Calendar c = new GregorianCalendar(year, month, day);
        return new SimpleDateFormat("EEEE").format(c.getTime());
    }
}
```

30

Notes:

The Remoting application begins with a service. Design a service based upon a clean, well-designed interface.

applicationContext.xml

```
<beans ... >
    <bean id="dayOfWeekService"
          class="com.company.springclass.service.DayOfWeekImpl"/>

    <bean id="serviceExporter"
          class="org.springframework.remoting.rmi.RmiServiceExporter">
           <property name="serviceName">
                  <value>DayOfWeek</value>
           </property>
           <property name="service">
                  <ref local="dayOfWeekService"/>
           </property>
           <property name="serviceInterface">
                  <value>com.company.springclass.service.DayOfWeek</value>
           </property>
           <property name="registryPort"><value>1099</value>
           </property>
           <property name="servicePort"><value>9226</value>
           </property>
    </bean>
</beans>
```

32

Notes:

The RmiServiceExporter's job is to create and register service properties for an RMI registry. The typical port for an RMI registry is 1099. This is the default if not specified. The service port is the port number that the RMI service will communicate on. The serviceName will be the name of the RMI registry entry for this service. The service property is very important as it defines the Spring bean to use for the RMI service.

The Client Code

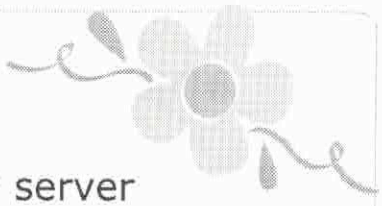
```
public class DayOfWeekClient {
    private DayOfWeek dayOfWeekService;
    public void setDayOfWeekService(DayOfWeek dayOfWeekService){
        this.dayOfWeekService = dayOfWeekService;
    }
    public static void main(String[] args) throws Exception {
        ApplicationContext ctx =
            new FileSystemXmlApplicationContext(
                "applicationContext-client.xml");
        DayOfWeekClient dayOfWeekClient =
            (DayOfWeekClient) ctx.getBean("dayOfWeekClient");
        dayOfWeekClient.run();
    }
    public void run() {
    System.out.println(dayOfWeekService.getDayOfWeek(2009, 0, 1));
    }
}
```

34

Notes:

This is the client code for our client-server application. Notice it has a reference and setter method to the service interface. In the *main()* method, once the app starts, the client's Spring IOC container is created. This causes the Spring config file to be read and the appropriate beans to be instantiated. As you can see on the next slide, the dayOfWeekService is injected by Spring into this application. Therefore, we instantiate the client and have at our disposal the dayOfWeekService as a result.

Summary

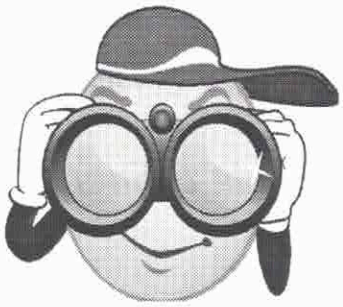


- Spring Remoting makes client / server development an easy task by removing much of the ugly coding of the proxy code on the server.
- Create separate Spring configurations for both the server and the client
- The client will need to configure an RmiProxyFactoryBean
- The server will need to define a ServiceExporter

36

Notes:

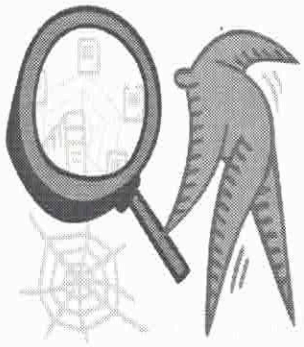
***This page intentionally left blank*!**



38

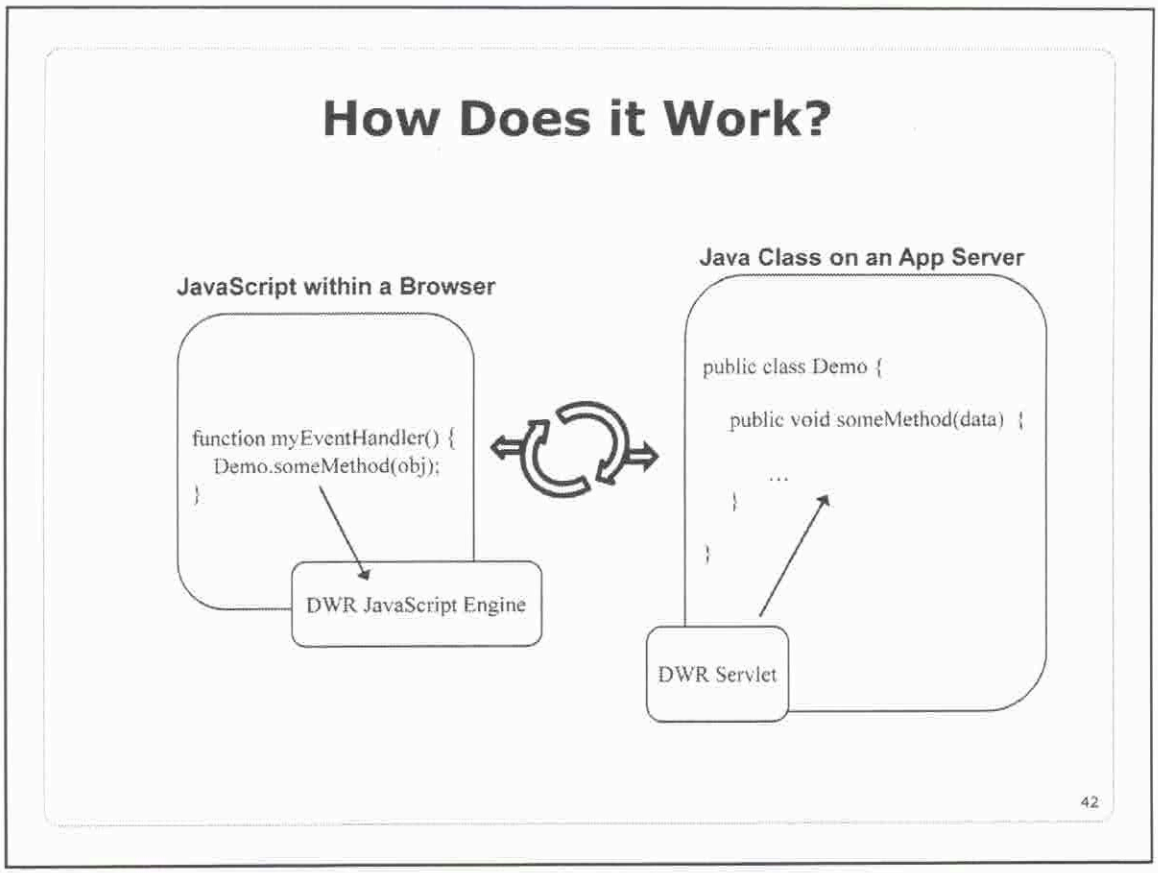
Spring and Web 2.0 Overview

- What is DWR?
- Spring DWR Integration



40

Notes:



Notes:

A developer writes a JavaScript function that appears to directly invoke a Java class and function on the server-side. In reality, the DWR JavaScript engine intercepts this function call, handles the Ajax communications to the DWR servlet, which invokes the method in the Java class. The DWR framework then passes the response back to the JavaScript Engine and invokes a developer-created callback in the JavaScript code.

Setting Up DWR ...

- Establish the DWR Servlet in web.xml:

```
<servlet>
        <servlet-name>dwr-invoker</servlet-name>
        <servlet-class>
          org.directwebremoting.servlet.DwrServlet
        </servlet-class>
        <init-param>
          <param-name>debug</param-name>
          <param-value>true</param-value>
        </init-param>
</servlet>

<servlet-mapping>
        <servlet-name>dwr-invoker</servlet-name>
        <url-pattern>/dwr/*</url-pattern>
</servlet-mapping>
```

To see DWR debug info, visit local:8080/SpringDWR01/dwr/index.html

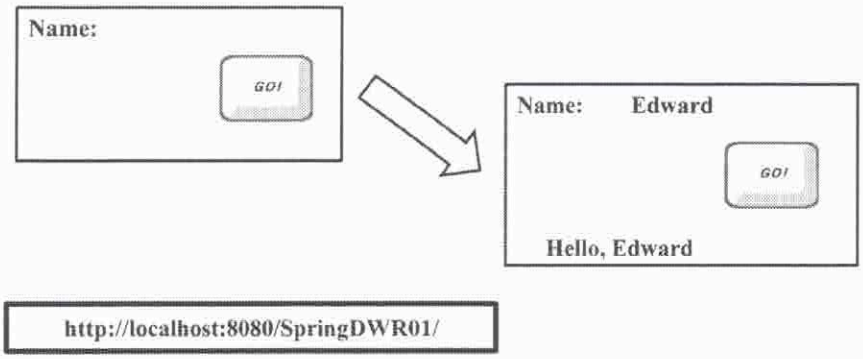
44

Notes:

Add the DWR servlet into your web.xml along with your desired mapping. The debug init-param allows for directory browsing of all configured DWR objects that are exposed.

Simple DWR Example

- The following example illustrates how to make a DWR call

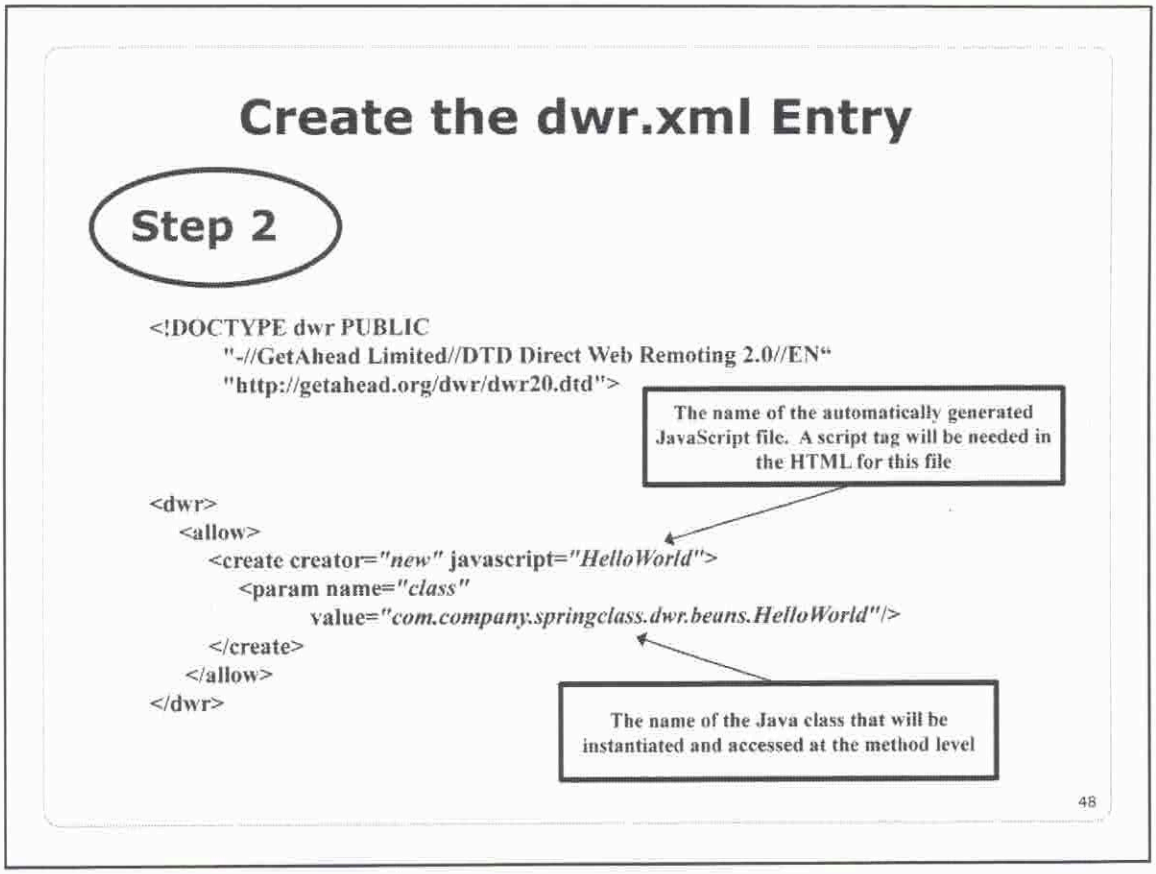


http://localhost:8080/SpringDWR01/

46

Notes:

In this example, we will create a form that submits data and gets a response from the server as shown using DWR. The URL to view this example (server must be running) is shown in the slide.



Notes:

This file identifies how DWR should create the JavaScript files and what files to allow access to.

The create element tells DWR that a server-side class should be exposed to Ajax requests and defines how DWR should obtain an instance of that class to remote. The creator attribute here is set to the value new, meaning that DWR should call the class's default constructor to obtain an instance. Other possibilities are to create an instance through a fragment of script using the Bean Scripting Framework (BSF), or to obtain an instance via integration with the IOC container, Spring. By default, when an Ajax request to DWR invokes a creator, the instantiated object is placed in page scope and therefore is no longer available after the request completes.

Create the HTML Page

Step 4

- The form invokes a JavaScript function:

```
<div id="wrapper">
   <form onsubmit="return handleForm();">
     <label for="name">Name:</label>
     <input type="text" id="name" name="name"></input>
      <input id="submit" type="submit" value="Go!"> </input>
   </form>
   <div id="resultsDiv"></div>
</div>
```

50

Notes:

Our HTML form invokes the *handleForm* JavaScript function.

Integrating Spring and DWR

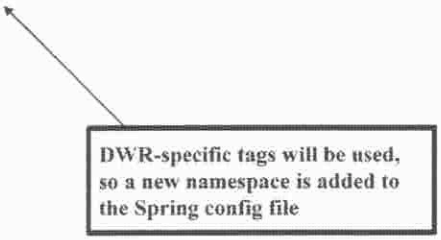
- DWR 2.0 allows for Spring to maintain all necessary DWR config info
  - *dwr.xml* can be removed
- DWR can invoke Spring services directly
  - Load a different DwrServlet
  - This one is Spring-aware
  - **org.directwebremoting.spring.DwrSpringServlet**

52

Notes:

applicationContext.xml

```
<beans    xmlns=http://www.springframework.org/schema/beans"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:dwr="http://www.directwebremoting.org/schema/spring-dwr"
          xsi:schemaLocation="http://www.springframework.org/schema/beans"
     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
     http://www.directwebremoting.org/schema/spring-dwr
     http://www.directwebremoting.org/schema/spring-dwr-3.0.xsd">
     ...

</beans>
```



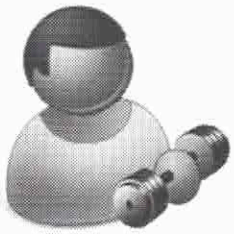
54

Notes:

The contents of the config file are on the next slide.

Lab 8: Spring and DWR

- Complete the application following the additional instructions in **SpringLab08**
- This exercise will use Spring and DWR and make use of the *EmployeeService*



56

Notes:

This page intentionally left blank!

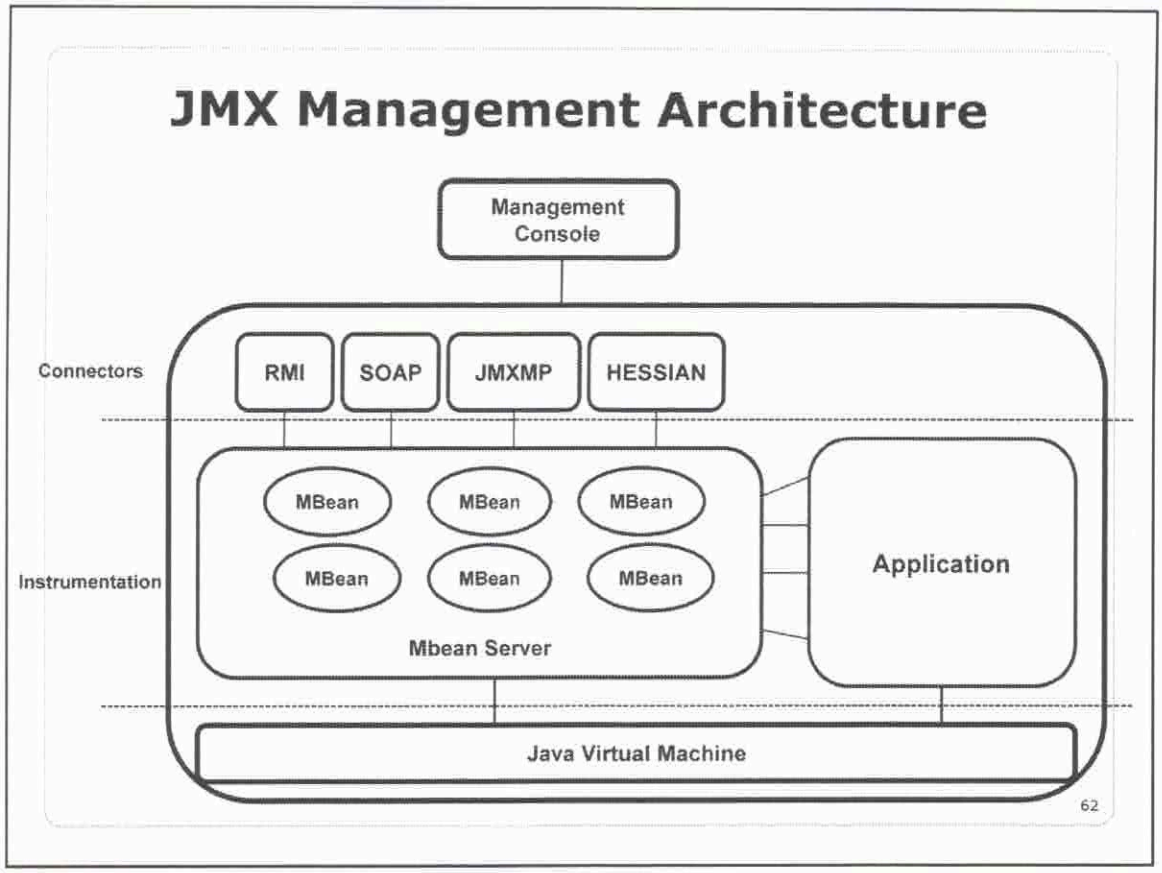
58

Overview

- Introducing JMX
  - JMX Basic Definition
  - MBean Registration
  - Spring and JMX

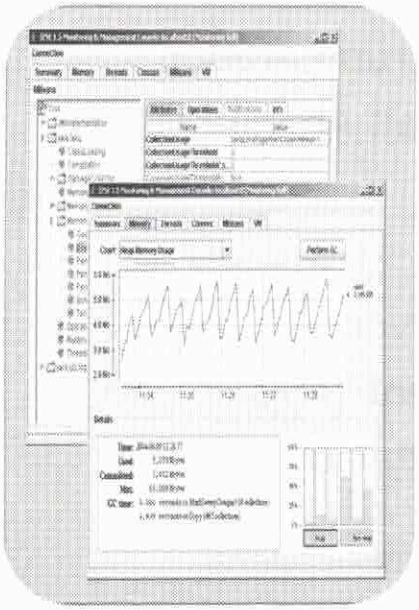
60

Notes:


JMX Management Architecture
Management Console
Connectors
RMI
SOAP
JMXMP
HESSIAN
Instrumentation
MBean
MBean
MBean
MBean
MBean
MBean
Mbean Server
Application
Java Virtual Machine
62

Notes:

JConsole



- Desktop Management Console
- Graphically Expose MBeans
- Part of Java 5
- Info on GC, Memory, Threads
- Support Remote Connection
- Built-in Security
- Requires VM-Argument Switch

64

Notes:

JMX Usage Pattern

- JVM Profiling, Monitoring & Control
- Application Monitoring
  - Exposing instrumented aspects
  - Monitoring of activities and dynamic values
  - Notifying or reacting to state changes
- Application Control
  - Control life cycle of application components
  - Expose settings as push-button controls
- Configuration
  - Runtime update of application state

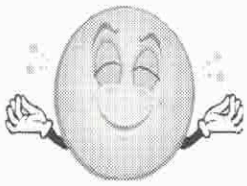
66

Notes:

Code Samples

- JVM command line parameter for jConsole

-Dcom.sun.management.jmxremote

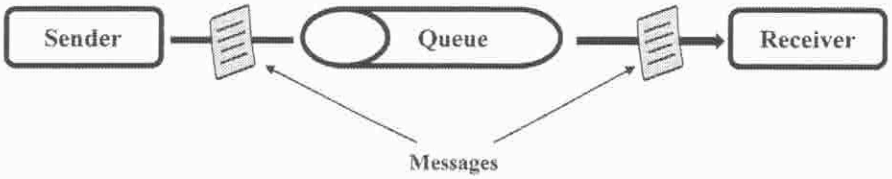


68

Notes:

JMS Overview

- Asynchronous Messaging
  - Brokers and Destinations
- Queues → One Sender / One Receiver
  - Message is removed from queue once received
  - Multiple receivers can listen to the same queue



70

Notes:

Benefits

- No waiting
- Service specifics are not required
- Location independence
- Guaranteed delivery

72

Notes:

Steps ...

- Declare a message destination
  - Queue

```
<bean id="rantzDestination"
      class="org.apache.activemq.command.ActiveMQQueue">
   <constructor-arg index="0" value="rantz.marketing.queue"/>
</bean>
```

  - Topic

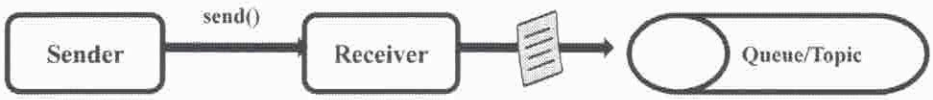
```
<bean id="rantzDestination"
      class="org.apache.activemq.command.ActiveMQTopic">
   <constructor-arg index="0" value="rantz.marketing.topic"/>
</bean>
```

74

Notes:

Sending Messages

- Sending a message, the Big Picture



76

Notes:

Sending Messages ...

```
public void sendMotoristInfo(final Motorist motorist) {
    jmsTemplate.send (                                          <-- Sends Message
        destination,                                            <-- Specifies destination
        new MessageCreator() {
          public Message createMessage(Session session)
              throws JMSException {                             <-- Creates message
            MapMessage message = session.createMapMessage();
            message.setString("lastName", motorist.getLastName());
            message.setString("firstName", motorist.getFirstName());
            message.setString("email", motorist.getEmail());

            return message;
          }
        });
}

private JmsTemplate jmsTemplate;                                <-- Injects JmsTemplate
public void setJmsTemplate(JmsTemplate jmsTemplate) {
    this.jmsTemplate = jmsTemplate;
}

private Destination destination;                                <-- Injects Destination
public void setDestination(Destination destination) {
    this.destination = destination) {
}
}
```

78

Notes:

Consuming a Message

- Receiving messages with JmsTemplate

80

Notes:

Consuming a Message ...

- The Code (cont'd)

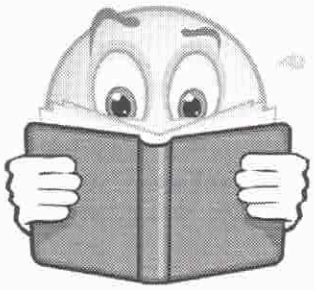
```
    } catch (JMSException e) {
        throw JmsUtils.convertJmsAccessException (e) ;
      }
      return motorist;
    }

    //injected
    private JmsTemplate jmsTemplate;
    public void setJmsTemplate(JmsTemplate jmsTemplate) {
      this.jmsTemplate = jmsTemplate;
    }
}
```

Converts any JMSException

82

Notes:

This page intentionally left blank!



84