JavaOne

# Java EE Connectors
## The Secret Weapon Reloaded


Jonathan Gallimore
Tomitribe

David Blevins
Tomitribe


JavaOne

# Inbound Connectors (aka MDBs)





@jongallimore @dblevins @tomitribe

JavaOne

# Dispelling Myths

# Which Statements are Always True?

- MDBs are for JMS
- MDBs must be asynchronous
- A MessageListener interface has one method which returns ‘void’
- MDBs are pooled
- MDBs are stateless

# The Truth

- MDBs are actually “Connector Driven Beans”
- MDBs are not asynchronous or synchronous
  - Connectors are asynchronous or synchronous
- A MessageListener interface may:
  - have any number of methods
  - have any return type
- MDBs are not pooled, stateful or stateless
  - The Connector has that choice

# The Promise

- Infinite Extensibility
- Push Communication
  - To any server
  - From anywhere
- Support any protocol
  - Optional transaction guarantees

JavaOne

# Did it deliver?

JavaOne

# Technically, yes

# What Spoiled It?

- JMS legacy confused and misleading
  - Pigeonholed thinking
- Specification too large
  - Outbound Connectors are very complex
- No strong second use-case
- Crippled configuration
  - Effectively a typeless map

JavaOne

# JMS MDB in Java EE 7





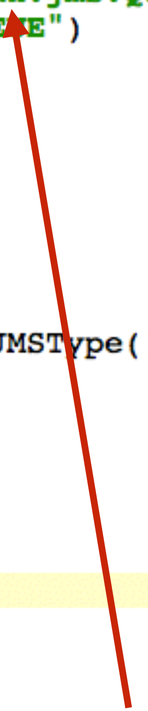
@jongallimore @dblevins @tomitribe

JavaOne

```
@MessageDriven(activationConfig = {
        @ActivationConfigProperty(propertyName = "maxSessions", propertyValue = "3"),
        @ActivationConfigProperty(propertyName = "maxMessagesPerSessions", propertyValue = "1"),
        @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Queue"),
        @ActivationConfigProperty(propertyName = "destination", propertyValue = "TASK.QUEUE")
})
public class BuildTaskMessageListener implements MessageListener {

    @Override
    public void onMessage(Message message) {

        try {
            if (!(message instanceof ObjectMessage)) {
                throw new JMSException("Expected ObjectMessage, received " + message.getJMSType());
            }

            final ObjectMessage objectMessage = (ObjectMessage) message;

            final BuildTask buildTask = (BuildTask) objectMessage.getObject();

            buildTask.doSomethingUseful();

        } catch (JMSException e) {
            // Why can't I throw a JMSException???
            throw new RuntimeException(e);
        }

    }
}
```

# User manual required…

```java
@MessageDriven(activationConfig = {
        @ActivationConfigProperty(propertyName = "maxSessions", propertyValue = "3"),
        @ActivationConfigProperty(propertyName = "maxMessagesPerSessions", propertyValue = "1"),
        @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Queue"),
        @ActivationConfigProperty(propertyName = "destination", propertyValue = "TASK.QUEUE")
})
public class BuildTaskMessageListener implements MessageListener {

    @Override
    public void onMessage(Message message) {

        try {
            if (!(message instanceof ObjectMessage)) {
                throw new JMSException("Expected ObjectMessage, received " + message.getJMSType());
            }

            final ObjectMessage objectMessage = (ObjectMessage) message;

            final BuildTask buildTask = (BuildTask) objectMessage.getObject();

            buildTask.doSomethingUseful();

        } catch (JMSException e) {
            // Why can't I throw a JMSException???
            throw new RuntimeException(e);
        }

    }
}
```



# Loosely typed

```
@MessageDriven(activationConfig = {
        @ActivationConfigProperty(propertyName = "maxSessions", propertyValue = "3"),
        @ActivationConfigProperty(propertyName = "maxMessagesPerSessions", propertyValue = "1"),
        @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Queue"),
        @ActivationConfigProperty(propertyName = "destination", propertyValue = "TASK.QUEUE")
})
public class BuildTaskMessageListener implements MessageListener {

    @Override
    public void onMessage(Message message) {

        try {
            if (!(message instanceof ObjectMessage)) {
                throw new JMSException("Expected ObjectMessage, received " + message.getJMSType());
            }

            final ObjectMessage objectMessage = (ObjectMessage) message;

            final BuildTask buildTask = (BuildTask) objectMessage.getObject();

            buildTask.doSomethingUseful();

        } catch (JMSException e) {
            // Why can't I throw a JMSException???
            throw new RuntimeException(e);
        }

    }
}
```

# Poor targeting

```java
@MessageDriven(activationConfig = {
        @ActivationConfigProperty(propertyName = "maxSessions", propertyValue = "3"),
        @ActivationConfigProperty(propertyName = "maxMessagesPerSessions", propertyValue = "1"),
        @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Queue"),
        @ActivationConfigProperty(propertyName = "destination", propertyValue = "TASK.QUEUE")
})
public class BuildTaskMessageListener implements MessageListener {

    @Override
    public void onMessage(Message message) {

        try {
            if (!(message instanceof ObjectMessage)) {
                throw new JMSException("Expected ObjectMessage, received " + message.getJMSType());
            }

            final ObjectMessage objectMessage = (ObjectMessage) message;

            final BuildTask buildTask = (BuildTask) objectMessage.getObject();

            buildTask.doSomethingUseful();

        } catch (JMSException e) {
            // Why can't I throw a JMSException???
            throw new RuntimeException(e);
        }

    }
}
```

# Static interface

# So 2004....

```
@MessageDriven(activationConfig = {
        @ActivationConfigProperty(propertyName = "maxSessions", propertyValue = "3"),
        @ActivationConfigProperty(propertyName = "maxMessagesPerSessions", propertyValue = "1"),
        @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.Queue"),
        @ActivationConfigProperty(propertyName = "destination", propertyValue = "TASK.QUEUE")
})
public class BuildTaskMessageListener implements MessageListener {

    @Override
    public void onMessage(Message message) {

        try {
            if (!(message instanceof ObjectMessage)) {
                throw new JMSException("Expected ObjectMessage, received " + message.getJMSType());
            }

            final ObjectMessage objectMessage = (ObjectMessage) message;

            final BuildTask buildTask = (BuildTask) objectMessage.getObject();

            buildTask.doSomethingUseful();

        } catch (JMSException e) {
            // Why can't I throw a JMSException???
            throw new RuntimeException(e);
        }

    }
}
```

# Learning from JAX-RS

- No interfaces
- Fluid method signatures
- Annotation-based configuration
  - Strongly Typed
  - Self-documenting
  - Targeted
    - Class-level
    - Method-level
    - Parameter-level

# 2 Simple Changes

- Give Connector the full Bean Class
  - Hello, Annotations!
  - Modern Expressive API design
- Allow Connector to invoke Any Bean Method
  - Bean Developer gains full freedom
  - APIs people would actually use and enjoy

# Oh yeah.

```
@MessageDriven
@MaxSessions(3)
@MaxMessagesPerSession(1)
public class BuildTaskMessageListener {

    @Destination(value = "TASK.QUEUE")
    @DestinationType(Queue.class)
    @MessageType(ObjectMessage.class)
    public void processBuildTask(BuildTask buildTask) throws JMSException {

        buildTask.doSomethingUseful();
    }

    @Destination(value = "BUILD.TOPIC")
    @DestinationType(Topic.class)
    @MessageType(ObjectMessage.class)
    public void process(BuildNotification notification) throws JMSException {

        System.out.println("Something happened: " + notification);

    }

}
```

JavaOne

# Beyond JMS





@jongallimore @dblevins @tomitribe

JavaOne

# Tutorial

# Writing an Inbound Connector

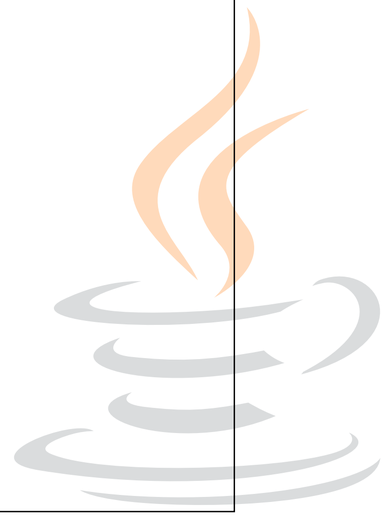
- ResourceAdapter
- MessageListener API
- ActivationSpec
- ra.xml

# Creating the resource adapter

- Implement javax.resource.spi.ResourceAdapter
  - provide metadata with @Connector
- Implement start() and stop() methods
  - perform the connector's logic (sockets, threads, etc)

# Creating the resource adapter

```
@Connector(description = "Twitter Resource Adapter",
           displayName = "Twitter Resource Adapter",
           eisType = "Twitter Resource Adapter", version = "1.0")
public class TwitterResourceAdapter implements ResourceAdapter, StatusChangeListener {

    private TwitterStreamingClient client;

    public void start(final BootstrapContext bootstrapContext)
            throws ResourceAdapterInternalException {
        client = new TwitterStreamingClient(this, consumerKey, consumerSecret,
                                            accessToken, accessTokenSecret);
        try {
            client.run();
        } catch (InterruptedException | ControlStreamException | IOException e) {
            // TODO: custom error handling
        }
    }

    public void stop() {
        client.stop();
    }
}
```

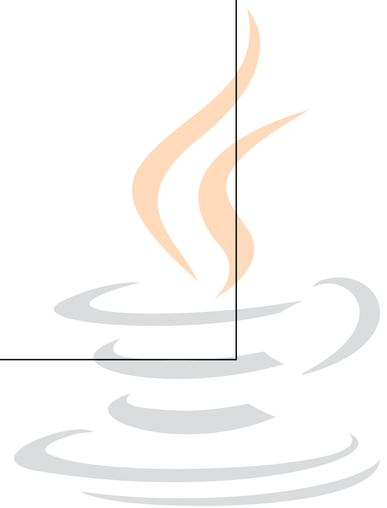


# Configuring the resource adapter

- Configuration can be provided using fields on the ResourceAdapter implementation
- Values can be provided in ra.xml, or through app server specific configuration mechanisms
- TomEE uses system properties
  - e.g. twitter-connector-rar-0.1RA.consumerKey = <consumer key>

# Configuring the resource adapter

```
@Connector(description = "Twitter Resource Adapter",
           displayName = "Twitter Resource Adapter",
           eisType = "Twitter Resource Adapter", version = "1.0")
public class TwitterResourceAdapter implements ResourceAdapter, StatusChangeListener {

    @ConfigProperty
    @NotNull
    private String consumerKey;

    @ConfigProperty
    @NotNull
    private String consumerSecret;
}
```



# Registering the MDB

- ResourceAdapter.endpointActivation(), store
  - ActivationSpec
  - MessageEndpoint
  - bean class
- ResourceAdapter. endpointDeactivation() , remove
  - ActivationSpec
  - MessageEndpoint.release()

# Registering the MDB

```
public void endpointActivation(final MessageEndpointFactory messageEndpointFactory,
    final ActivationSpec activationSpec) throws ResourceException {

    final TwitterActivationSpec twitterActivationSpec = (TwitterActivationSpec)
activationSpec;
    final MessageEndpoint messageEndpoint = messageEndpointFactory.createEndpoint(null);

    final Class<?> endpointClass = twitterActivationSpec.getBeanClass() != null ?
twitterActivationSpec
            .getBeanClass() : messageEndpointFactory.getEndpointClass();

    final EndpointTarget target = new EndpointTarget(messageEndpoint, endpointClass);
    targets.put(twitterActivationSpec, target);
}
```

# Invoking the MDB

- Can be done at any time in the connector
- Call Method.invoke() with the MessageEndpoint as the object reference
- Wrap with messageEndpoint.beforeDelivery() and messageEndpoint.afterDelivery()

```
public void invokeEndpoints() {
    try {
        messageEndpoint.beforeDelivery(method);
        final Object[] values = … // values to pass in as parameters
        method.invoke(messageEndpoint, values);
    } finally {
        messageEndpoint.afterDelivery();
    }
}
```

# Packaging the connector

- Typically delivered in a .rar file
  - Installed in the container
  - Or part of an EAR file
- Package connector API files separately from the implementation

# Sounds like a lot…

- Help is here!
  - Connector starter projects
    - https://github.com/tomitribe/connector-starter-project
    - https://github.com/tomitribe/connector-starter-project-with-security
    - https://github.com/tomitribe/connector-starter-project-inbound
    - https://github.com/tomitribe/connector-starter-project-outbound
- Examples
  - https://tomitribe.io/projects/chatterbox
  - https://tomitribe.io/projects/sheldon

JavaOne

# Sample Connectors

# Example: IMAP & Twitter

- Inbound connector with threads to carry out the actual work
- Uses reflection with annotations to filter messages for different methods on the MDB
  - Note - no @ActivationConfig
- Strongly typed method parameters
- JAX-RS style parameters (modelled on @Path / @PathParam)

JavaOne

# Advanced Inbound





@jongallimore @dblevins @tomitribe

# WorkManager

- Connectors provide a WorkManager
  - Manages a configurable number of threads
  - Submitting work can be blocking or non-blocking
  - Work object is essentially like a Runnable

```
@Connector(description = “My Connector")
public class MyResourceAdapter implements ResourceAdapter {

    public void start(BootstrapContext bootstrapContext) throws ResourceAdapterInternalException {
        workManager = bootstrapContext.getWorkManager();
    }

    public void start(BootstrapContext bootstrapContext) throws ResourceAdapterInternalException {
       workManager.scheduleWork(new Work() {

            @Override
            public void run() {
                // TODO: do some work here
            }
        });
    }
}
```

# Security

- Invoke a MDB in the context of an authenticated user
- Create a class that extends SecurityContext

```
public class WorkSecurityContext extends SecurityContext {

    @Override
    public void setupSecurityContext(final CallbackHandler handler,
        final Subject executionSubject, final Subject serviceSubject) {

        List<Callback> callbacks = new ArrayList<Callback>();

        final PasswordValidationCallback pvc = new PasswordValidationCallback(executionSubject, username,
            password.toCharArray());

        callbacks.add(pvc);

        Callback callbackArray[] = new Callback[callbacks.size()];
        try {
            handler.handle(callbacks.toArray(callbackArray));
        } catch (UnsupportedCallbackException e) {
            // handle this
        }

        this.authenticated = pvc.getResult();
    }
}
```

# Security continued

- Add custom SecurityContext to Work Object
- The server will process the callbacks and run the work as the required principal

```
public class AuthenticateWork implements Work, WorkContextProvider {

   private final WorkSecurityContext securityContext = new WorkSecurityContext();

    @Override
    public void run() {
        // do work here
    }

    @Override
    public List<WorkContext> getWorkContexts() {
        return Collections.singletonList((WorkContext) securityContext);
    }
}
```

# Example: Sheldon

- Inbound connector
- Demonstrates security and work
- SSH session uses CREST to call @Commands on MDBs
- Easy to add commands

```
@MessageDriven(name = "Echo")
public class UserBean implements CommandListener {

    @Command
    public String echo(final String input) {
        return input;
    }
}
```

JavaOne

# Outbound Connectors (harder)





@jongallimore @dblevins @tomitribe

# Creating an Outbound connector

- API
  - ConnectionFactory
  - Connection
- Implementation
  - ManagedConnectionFactory
  - ConnectionFactoryImpl
  - ManagedConnection
  - ConnectionImpl
- Complex, but **mostly boilerplate** for simple cases

# ConnectionFactory

- Application component interacts with the ConnectionFactory and Connection
- ManagedConnectionFactory and ManagedConnection hide complexities from the application

```
public class XMPPManagedConnectionFactory implements ManagedConnectionFactory, ResourceAdapterAssociation {

    public Object createConnectionFactory(ConnectionManager cxManager) throws ResourceException {
        return new XMPPConnectionFactoryImpl(this, cxManager);
    }

    public Object createConnectionFactory() throws ResourceException {
        throw new ResourceException("This resource adapter doesn't support non-managed environments");
    }

    public ManagedConnection createManagedConnection(Subject subject, ConnectionRequestInfo cxRequestInfo) throws ResourceException {
        return new XMPPManagedConnection(this);
    }

    public ManagedConnection matchManagedConnections(Set connectionSet,
                                                     Subject subject, ConnectionRequestInfo cxRequestInfo) throws ResourceException {
        ManagedConnection result = null;
        Iterator it = connectionSet.iterator();
        while (result == null && it.hasNext()) {
            ManagedConnection mc = (ManagedConnection) it.next();
            if (mc instanceof XMPPManagedConnection) {
                result = mc;
            }

        }
        return result;
    }
}
```

# ConnectionFactory continued

- Connection factory calls the ConnectionManager to obtain a connection from the ManagedConnectionFactory

```
public class XMPPConnectionFactoryImpl implements XMPPConnectionFactory {
    private static final long serialVersionUID = 1L;

    private Reference reference;
    private XMPPManagedConnectionFactory mcf;
    private ConnectionManager connectionManager;

    public XMPPConnectionFactoryImpl(XMPPManagedConnectionFactory mcf, ConnectionManager cxManager) {
        this.mcf = mcf;
        this.connectionManager = cxManager;
    }

    @Override
    public XMPPConnection getConnection() throws ResourceException {
        log.finest("getConnection()");
        return (XMPPConnection) connectionManager.allocateConnection(mcf, null);
    }
}
```

# Connection

- ManagedConnectionFactory creates ManagedConnection objects
- Add methods to the connection implementation to call the external service

```
public class XMPPManagedConnection implements ManagedConnection {

    private XMPPManagedConnectionFactory mcf;
    private List<ConnectionEventListener> listeners;
    private XMPPConnectionImpl connection;

    public XMPPManagedConnection(XMPPManagedConnectionFactory mcf) {
        this.mcf = mcf;
        this.logwriter = null;
        this.listeners = Collections.synchronizedList(new ArrayList<ConnectionEventListener>(1));
        this.connection = null;
    }

    public Object getConnection(Subject subject,
                                ConnectionRequestInfo cxRequestInfo) throws ResourceException {
        log.finest("getConnection()");
        connection = new XMPPConnectionImpl(this, mcf);
        return connection;
    }

    public void sendMessage(String recipient, String message) throws MessageException {
        log.finest("sendMessage()");

        final XMPPResourceAdapter resourceAdapter = (XMPPResourceAdapter) mcf.getResourceAdapter();
        resourceAdapter.sendXMPPMessage(recipient, message);
    }
}
```

# Using a connection

- Use @Resource to inject the connection factory into a managed component
- Obtain a connection from the connection factory and use it
- Do not forget to close the connection when you have finished

```
@MessageDriven(name = "Chat")
public class ChatBean implements XMPPMessageListener {

    @Resource
    private XMPPConnectionFactory cf;

    @MessageText("echo {message:.*$}")
    public void echo(@SenderParam final String sender, @MessageTextParam("message") final String message)
throws Exception {
        final XMPPConnection connection = cf.getConnection();
        connection.sendMessage(sender, message);
        connection.close();
    }
}
```

# Example: XMPP

- Bi-directional connector
- Incoming messages call MDBs
- Application components can use the ConnectionFactory to send messages
- MDB uses the ConnectionFactory to create an “interactive bot”

JavaOne

# Want more?

- Examples
  - https://tomitribe.io/projects/chatterbox
  - https://github.com/tomitribe/connector-starter-project

- Standardized Extension-Building in Java EE with CDI and JCA [CON2385], Jason Porter, Oct 28th, 4.30pm, Parc 55 Mission

JavaOne

# Thank You!


**Jonathan Gallimore**
Tomitribe

**David Blevins**
Tomitribe