dispatches from the codeface

SUNDAY, 3 MARCH 2013

Spring ActiveMQ Producer Client Template

AVAILABLE ON GITHUB

Demo available on GitHub

OVERVIEW

This is just a very small, very simple template project for sending messages to JMS queues.

In my current role we have a lot of different components and systems glued together by ActiveMQ and Camel. When developing\integration testing in this environment - it is often useful to create ad-hoc messages and send them to queues directly. This is usually the case when you need to test a subset of a larger system.

I usually knock up a test app to do this, often replicating this functionality time and time again. So I thought it might be useful to pop this into a simple template project that can be easily reused. It also made sense to 'mavenise' and 'springify' to make it more easily extensible if you need to plug in other components and features in tests.

Just to reiterate - this is just a starting point for developer style, ad hoc testing to support the development process.

RUNNING THE DEMO

1. Prerequisites
    1. Running instance of ActiveMQ (local/remote)
    2. Maven and JDK installed
2. Configuring
    1. Edit application.properties to point to your ActiveMQ instance - specifying broker url and queue name
    2. Edit 'StartDemo.java' to provide the ObjectMessage the endpoint is expecting
    3. Type 'maven exec:java' to run the StartDemo class

application.properties

```
broker.url=tcp://localhost:61616
broker.queue=MyTestQueue
```

ActiveMQ - application.properties hosted with ♥ by GitHub view raw

LOOKING AT THE CODE

All the Spring configuration is set up in the main application-context.xml

application-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:jms="http://www.sprin
        xmlns:context="http://www.springframework.org/schema/context"
        xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.spring
                    http://www.springframework.org/schema/jms http://www.springframew
                                        http://www.springframework.org/schema/cor

        <!-- Process Spring Annotations -->
        <context:component-scan base-package="com.cor.demo" />
```

BLOG ARCHIVE

► 2014 (2)

▼ 2013 (7)

► August (2)

► July (1)

► May (1)

► April (1)

▼ March (1)

Spring ActiveMQ Producer Client Template

► February (1)

ABOUT ME

Adrian Milne

Java enterprise developer by day, mobile app and new tech explorer by night..

View my complete profile

POPULAR POSTS

Analysing REST Web Services with Squid and AWStats

Background I've got a Java application running on Tomcat on Amazon EC2 , which exposes RESTful web services delivering JSON to a varie...

Complex Event Processing Made Easy (using Esper)

The following is a very simple example of event stream processing (using the ESPER engine). Note - a full working example is availabl...

502 Proxy Error Using CometD, Apache and Camel

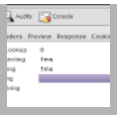
```
11
12          <bean id="serverProperties" class="org.springframework.beans.factory.config.Prope
13                  <property name="systemPropertiesModeName" value="SYSTEM_PROPERTIES_MODE_
14                  <property name="placeholderPrefix" value="${applicationProperties."/>
15                  <property name="locations">
16                  <list>
17                          <value>classpath:/application.properties</value>
18                  </list>
19              </property>
20                  <property name="ignoreResourceNotFound" value="true"/>
21          </bean>
22
23          <!-- A JMS connection factory for ActiveMQ -->
24          <bean id="connectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory
25                  p:brokerURL="${applicationProperties.broker.url}" />
26
27
28          <!-- A destination in ActiveMQ -->
29          <bean id="destination" class="org.apache.activemq.command.ActiveMQQueue">
30                  <constructor-arg value="${applicationProperties.broker.queue}" />
31          </bean>
32
33          <!-- A cached connection to wrap the ActiveMQ connection -->
34          <bean id="cachedConnectionFactory"
35                  class="org.springframework.jms.connection.CachingConnectionFactory"
36                  p:targetConnectionFactory-ref="connectionFactory" p:sessionCacheSize="10"
37
38          <!-- A JmsTemplate instance that uses the cached connection and destination -->
39          <bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate"
40                  p:connectionFactory-ref="cachedConnectionFactory"
41                  p:defaultDestination-ref="destination" />
42
43  </beans>
```

ActiveMQ - application-context.xml hosted with ♥ by **GitHub** **view raw**

The ObjectMessages are created in the main StartDemo.java class

StartDemo.java

```
1   package com.cor.demo.jms;
2
3   import java.io.Serializable;
4
5   import org.springframework.beans.factory.BeanFactory;
6   import org.springframework.context.ApplicationContext;
7   import org.springframework.context.support.ClassPathXmlApplicationContext;
8
9   import com.cor.demo.jms.dispatcher.MessageDispatcher;
10
11  /**
12  * Just a very simple template demo that will send a single Object Message to the JMS Brok
13  * Queue name defined in application.properties file. This is just for simple throwaway te
14  * purposes - designed as a starting point to develop on a case by case basis.
15  */
16  public class StartDemo {
17
18      /**
19  * @param args
20  */
21      public static void main(String[] args) {
22
23          // Load spring config
24          ApplicationContext appContext = new ClassPathXmlApplicationContext(new String[] {
25          BeanFactory factory = (BeanFactory) appContext;
26
27          // Send Message
28          MessageDispatcher dispatcher = (MessageDispatcher) factory.getBean("messageDispat
29          dispatcher.sendMessageObject(getTestMessage());
30      }
31
32      /**
33  * Generates the test message - overwrite with your own message.
34  * @return The object message to be sent to the JMS Queue
35  */
36      private static Serializable getTestMessage() {
37
38          return "MY TEST MESSAGE";
```



We recently hit an issue during testing with Apache returning "502 Proxy Errors" to clients who were connecting via CometD (using ...

JPA Searching Using Lucene - A Working Example with Spring and DBUnit

Working Example on Github There's a small, self contained mavenised example project over on Github to accompany this post - c...

MongoDB in 3 Easy Steps!

If you're entirely new to MongoDB - this is just a very gentle introduction that gets you up and running with a sample database in just...

CometD and Camel in the Enterprise - A Working Example

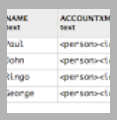
Full code is available on GitHub All the Java code for the project is available in Github - requires Maven to run. See the READM...

Spring ActiveMQ Producer Client Template

Available on GitHub Demo available on GitHub Overview This is just a very small, very simple template project for sending messages to ...

Data Pumper - Migrating Data from MySQL to PostgreSQL

I recently made the decision to migrate from MySQL to PostgreSQL (the reasons why to be covered later). I thought I'd just jot down my n...



Querying XML CLOB Data Directly in Oracle

Oracle 9 i introduced a new datatype - XMLType- specifically designed for handling XML naively in the database. However, we may find we h...

```
39      }
40  }
```

ActiveMQ - StarDemo.java hosted with ♥ by GitHub view raw

Communication with the server is abstracted away into a Dispatcher object.

MessageDispatcher.java

```
1  package com.cor.demo.jms.dispatcher;
2
3  import java.io.Serializable;
4
5  import javax.jms.JMSException;
6  import javax.jms.Message;
7  import javax.jms.ObjectMessage;
8  import javax.jms.Session;
9
10 import org.slf4j.Logger;
11 import org.slf4j.LoggerFactory;
12 import org.springframework.beans.factory.annotation.Autowired;
13 import org.springframework.jms.core.JmsTemplate;
14 import org.springframework.jms.core.MessageCreator;
15 import org.springframework.stereotype.Component;
16
17 /**
18  * Simple JMS Client - configured in activemq-client-context.xml.
19  * @author adrian.milne
20  *
21  */
22 @Component
23 public class MessageDispatcher {
24
25     /** Logger. */
26     private static Logger LOG = LoggerFactory.getLogger(MessageDispatcher.class);
27
28     /** JMS Template. */
29     @Autowired
30     protected JmsTemplate jmsTemplate;
31
32     /**
33      * Send the objectMessage to the Broker and Queue defined in application.properties.
34      * @param objectMessage Object Message
35      */
36     public void sendMessageObject(final Serializable objectMessage) {
37
38         LOG.info("Sending message " + objectMessage);
39
40         jmsTemplate.send(new MessageCreator() {
41
42             public Message createMessage(Session session) throws JMSException {
43                 ObjectMessage message = session.createObjectMessage(objectMessage);
44                 return message;
45             }
46         });
47
48         LOG.info("Message Sent!");
49
50     }
51
52 }
```

ActiveMQ - MessageDispatcher.java hosted with ♥ by GitHub view raw

And that's about it really!

CONCLUSION

As stated at the start - this is just a throw away project I use as a starting point when I need to write something to send messages to an ActiveMQ Queue/

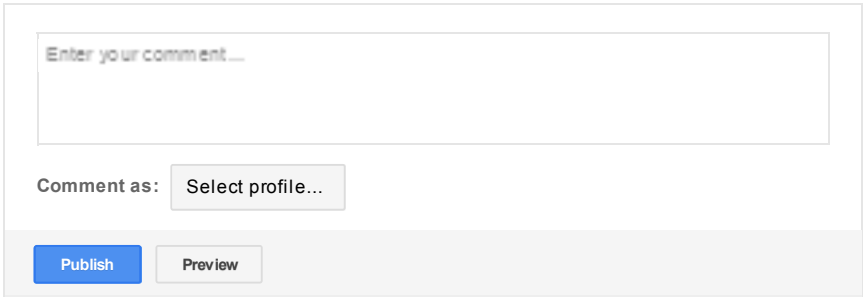
It's a compromise between something super simple (like a simple class), and something that lends itself easily to being extended. What can often start off as a simple test can grow into something more complex - and as we have maven and spring all configured at the start - it's then very easy

to add in extra dependencies if you find you need them.

Posted by Adrian Milne at 07:45

No comments:

Post a Comment



Newer Post Home Older Post

Subscribe to: Post Comments (Atom)