Spring Integration Control Bus Example

1. Introduction

Control bus is a useful Spring Integration component that accepts messages on the input channel similar to `Service Activator`, `Adapter` or `Transformer` but the key difference is that the payload of the message that is received indicates an invocable action or operation on a bean. The input channel is more of an operation channel that is basically used for sending control messages to invoke management operations on endpoints or other manageable resources.

Invocation of these operations are not just limited to Spring Integration's API classes alone, but user can define annotations anywhere in the code and then invoke these annotated methods using messages on control bus. We can specify the command messages to be sent to the control bus as simple `SpEL` expression and spring integration's core namespace provides implementation that evaluates the expression.

2. Control messages specification

In order for us to enable control bus, we have to add the following element to our configuration file as below.

```
1  <control-bus input-channel="testControlChannel"/>
```

The message sent to the control bus input channel should contain `SpEL` expression, referencing the bean and the control operation to be invoked on the bean. The "@" symbol is used to reference the bean. The operation to be invoked on the bean is specified with the operation name after the bean name. In the below example we will see how we can specify a simple control message to increment a sample counter on a managed resource bean.

```
1  Message incrementCounterMsg =
   MessageBuilder.withPayload(                    "@sampleCounter.increment()"
   ).build();
```

```
2  testControlChannel.send(incrementCounterMsg);
```

In the above case the `sampleCounter` bean has to be declared with annotation as `@ManagedResource` bean and the increment method has to be declared as `@ManagedOperation` for the control bus to interpret and invoke the operation.

Control bus can also be used to manage spring integration components. For example we can invoke operation on methods defined on `LifeCycle` interface even if the `@ManagedResource` annotation is not present. Adapters can be started or stopped on demand using control message. A simple example is shown as below

```
1  Message controlMessage =
   MessageBuilder.withPayload(                "@inboundAdapter.stop()"
   ).build(); testControlChannel.send(controlMessage);
```

Here in the above example we are sending a control message, to stop the bean `InboundAdapter` defined as part of the spring context as below.

```
1  <int:inbound-channel-adapter id="inboundAdapter" channel="msgChannelAdapterOutput"
             "'Sample
   expression=message'"          auto-startup="false">

2     <int:poller fixed-rate="1000"/>

3  </int:inbound-channel-adapter>
```

Let's go through an example of configuring a basic application for control bus demonstration.

3. Maven Dependencies

Below `pom.xml` shows the basic dependencies for configuring control bus. Spring integration core and spring integration jmx are the core dependencies

```
01  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi=
    "http://www.w3.org/2001/XMLSchema-instance"

02    xsi:schemaLocation=
    "http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">

03    <modelVersion>4.0.0</modelVersion>

04    <groupId>com.springinteg.controlbus</groupId>

05    <artifactId>spring-integration-controlbus</artifactId>

06    <packaging>war</packaging>

07    <version>1.0-SNAPSHOT</version>

08         >spring-integration-controlbus Maven
      <nameWebapp</                                    name>

09    <url>http://maven.apache.org</url>

10      <properties>

11          <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

12          <springframework.version>4.3.0.RELEASE</springframework.version>

13          <spring.integration.version>4.3.0.RELEASE</spring.integration.version>

14      </properties>
```

```
15    <dependencies>
16
17          <dependency>
18              <groupId>org.springframework.integration</groupId>
19              <artifactId>spring-integration-core</artifactId>
20              <version>${spring.integration.version}</version>
21          </dependency>
22          <dependency>
23               <groupId>org.springframework.integration</groupId>
24               <artifactId>spring-integration-jmx</artifactId>
25               <version>${spring.integration.version}</version>
26          </dependency>
27          <dependency>
28              <groupId>org.springframework.integration</groupId>
29              <artifactId>spring-integration-stream</artifactId>
30              <scope>compile</scope>
31              <version>${spring.integration.version}</version>
32          </dependency>
33
34          <dependency>
35              <groupId>org.springframework</groupId>
36              <artifactId>spring-test</artifactId>
37              <scope>test</scope>
```

```
38              <version>${spring.integration.version}</version>
39          </dependency>
40          <dependency>
41              <groupId>org.springframework.integration</groupId>
42              <artifactId>spring-integration-test</artifactId>
43              <scope>test</scope>
44              <version>${spring.integration.version}</version>
45          </dependency>
46          <dependency>
47              <groupId>junit</groupId>
48              <artifactId>junit</artifactId>
49              <version>3.8.1</version>
50              <scope>test</scope>
51          </dependency>
52          <dependency>
53        <groupId>log4j</groupId>
54        <artifactId>log4j</artifactId>
55        <version>1.2.17</version>
56        <scope>compile</scope>
57      </dependency>
58    </dependencies>
59    <build>
60      <finalName>spring-integration-controlbus</finalName>
```

```
61    </build>
62  </project>
```

4. Spring Integration Configuration

The core components that are part of control bus configuration are JMX Managed resource bean, inbound adapter, control-bus component and message channel. To expose the `ManagedResourceBean` for JMX monitoring and management, we need to export those attributes and operations. This can be done using the tag

```
1  <context:mbean-export/>
```

The detailed context configuration file `spring-integ-context.xml` with different components shown below.

```
01  <?xml version="1.0" encoding="UTF-8"?>
02  <beans xmlns="http://www.springframework.org/schema/beans"
03      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:int=
    "http://www.springframework.org/schema/integration"
04      xmlns:context="http://www.springframework.org/schema/context"
05
    xsi:schemaLocation="http://www.springframework.org/schema/integration
    http://www.springframework.org/schema/integration/spring-integration-4.3.xsd
06
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
07
    http://www.springframework.org/schema/integration/jmx
    http://www.springframework.org/schema/integration/jmx/spring-integration-jmx.xsd
08
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd">
09
10
11      <context:mbean-export />
12      <context:component-scan base-package="com.springinteg.controlbus" />
13
```

```
14      <int:channel id="msgChannelAdapterOutput">
15          <int:queue />
16      </int:channel>
17
18      <int:channel id="controlChannel" />
19
20      <int:control-bus input-channel="controlChannel" />
21      <int:inbound-channel-adapter id="inboundAdapter"
22                                                  "'This is a test
            channel="msgChannelAdapterOutput" expression=message'"
23          auto-startup="false">
24          <int:poller fixed-rate="1000" />
25      </int:inbound-channel-adapter>
26      <bean id="managedCounterBean" class="com.springinteg.controlbus.ManagedCounterBean" />
27  </beans>
```

5. Application Configuration

As you have noticed in the above spring context configuration we have defined `ManagedCounterBean` as a managed resource bean and also a control bus to invoke operations on this managed resource bean. The control bus listens on `controlChannel` for control messages and then invoke corresponding operations on managed attribute beans. Let's have a look at the implementation of these classes in detail below.

5.1 Managed Resource Bean

ManagedCounterBean.java

```
01  package com.springinteg.controlbus;
02  import java.util.concurrent.atomic.AtomicInteger;
03
04  import org.springframework.jmx.export.annotation.ManagedAttribute;
```

```
05 import org.springframework.jmx.export.annotation.ManagedOperation;
06 import org.springframework.jmx.export.annotation.ManagedResource;
07
08 @ManagedResource
09             ManagedCounterBean
   public class {
10                   AtomicInteger counter
       private final =                       new AtomicInteger();
11
12     @ManagedAttribute
13                getCounter()
       public int {
14         return this.counter.get();
15     }
16
17     @ManagedAttribute
18                                 counter)
       public void setCounter(int {
19         this.counter.set(counter);
20     }
21
22     @ManagedOperation
23                increment()
       public void {
24         this.counter.incrementAndGet();
25     }
```

```
26
27 }
```

6. Verification Test Configuration

The below code shows the basic test for verifying the control bus executing command messages sent on the message channel. The first test demonstrates how the command messages can control the operation defined on Spring's `LifecycleInterface`. In the first test before sending the control messages we assert that there are no messages in the channel. After sending the control message we make sure that inbound adapter is started and test messages are received by inbound adapter. The second test demonstrates how control messages can be used to invoke operations on a JMX `ManagedResourceBean` to increment a numeric counter.

ControlBusUnitTest.java

```
01 import static org.junit.Assert.*;
02
03 import org.apache.log4j.Logger;
04 import org.junit.Test;
05
06 import org.springframework.context.ConfigurableApplicationContext;
07 import org.springframework.context.support.ClassPathXmlApplicationContext;
08 import org.springframework.messaging.Message;
09 import org.springframework.messaging.MessageChannel;
10 import org.springframework.messaging.PollableChannel;
11 import org.springframework.messaging.support.GenericMessage;
12
13 import com.springinteg.controlbus.ManagedCounterBean;
14
15              ControlBusUnitTest
   public class {
16
```

```
17                  Logger logger =
    private static Logger.getLogger(ControlBusUnitTest.                class);
18
19      @Test
20                testControlbusAdapter()
    public void {
21          ConfigurableApplicationContext ac
            =                                  new ClassPathXmlApplicationContext(
"spring-integ-context.xml");
22          MessageChannel controlChannel =                                ,
            ac.getBean(                                "controlChannel"MessageChannel.  class
);
23          PollableChannel msgChannelAdapterOutput =
            ac.getBean(                                          "msgChannelAdapterOutput"
,
PollableChannel.  class);
24          Message receivedMsg = (Message)
            msgChannelAdapterOutput.receive(                                  1000);
25          assertNull(receivedMsg);
26                      "Message received on channel before adapter started:  +
            logger.info("                                                      receivedMsg);
27          controlChannel.send(new GenericMessage("@inboundAdapter.start()"));
28          receivedMsg = (Message)
            msgChannelAdapterOutput.receive(                          1000);
29          assertNotNull(receivedMsg);
30                      "Message received on channel adapter started:  +
            logger.info("                                              receivedMsg);
31          controlChannel.send(new GenericMessage("@inboundAdapter.stop()"));
32          receivedMsg = (Message)
            msgChannelAdapterOutput.receive(                          1000);
33          assertNull(receivedMsg);
```

```
34                      "Message received on channel after adapter stopped:  +
          logger.info("                                                        receivedMsg);
35          ac.close();
36      }
37
38      @Test
39                  testControlBusMBean()
        public void {
40          ConfigurableApplicationContext ac
            =                                  new ClassPathXmlApplicationContext(
    "spring-integ-context.xml");
41          MessageChannel controlChannel =                                  ,
            ac.getBean(                                "controlChannel"MessageChannel.  class
    );
42          ManagedCounterBean mangedCounterBean =
            ac.getBean(                                        "managedCounterBean"
    ,
    ManagedCounterBean.  class);
43          assertEquals(mangedCounterBean.getCounter(),0);
44                      "Value of message counter before sending message to control bus
          logger.info("
    +
    mangedCounterBean.getCounter());
45          controlChannel.send(new GenericMessage("@managedCounterBean.increment()"));
46          assertEquals(mangedCounterBean.getCounter(),1);
47                      "Value of message counter after sending message to control bus
          logger.info("
    +
    mangedCounterBean.getCounter());
48          ac.close();
49
50      }
```

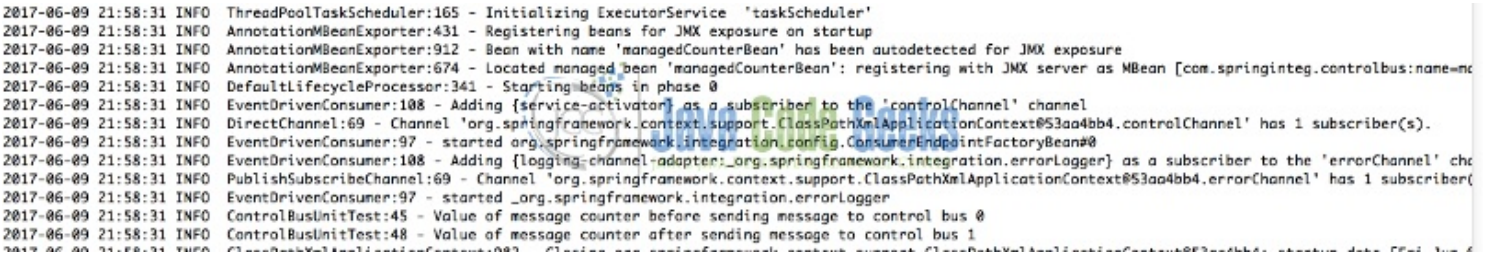
```
51

52  }
```

6.1 Screenshots of Test Execution

6.1.1 Adapter start and stop using command messages

Lifecycle operation control using command messages

6.1.2 Managed attribute increment using control messages



Managed Attribute control using command messages

7. Conclusion

In the above article we have looked at how control bus and command messages can help us in monitoring and management aspects of a messaging application. Monitoring and management is one of the critical aspects of a successful enterprise integration. Control bus can control different aspects of application through JMX operation. Further control messages can be specified using `SpEL` or `Groovy` for invoking operations on the target bean.

8. Download the source code

The source code for Spring Integration Control bus is as below.

Download
You can download the full source code of this example here: **spring-integration-controlbus**