Berner
Fachhochschule

Spring JMS

Stephan Fischli

Spring JMS Support

- Spring Boot automatically creates a JMS ConnectionFactory and the bean `JmsTemplate` which simplifies synchronous JMS messaging
- To receive messages asynchronously, any method of a managed bean can be annotated with `@JmsListener`
- To trigger the discovery of such methods, a configuration class must be annotated with `@EnableJms`

```
@SpringBootApplication
@EnableJms
public class Application {

    @Autowired
    private JmsTemplate jmsTemplate;

    @JmsListener(...)
    public void onMessage(Message message) {
        ...
```

```
        }
}
```

Spring JMS Support (cont.)

- By default, Spring Boot uses queues, but to use topics the property `spring.jms.pub-sub-domain` can be set
- When using ActiveMQ as a message broker, its URL can be configured via the property `spring.activemq.broker-url`

```
spring.jms.pub-sub-domain=true
spring.activemq.broker-url=tcp://localhost:61616
```

Sending Messages

- A message can be sent to a destination using the `send` method of the Spring JMS template and by providing a message creator

```
jmsTemplate.send(destination, session -> {
    Message message = session.createTextMessage(text);
    message.setDoubleProperty("version", 2.5);
    message.setText(...);
    return message;
});
```

- If the message body can be converted by a configured message converter, the message can be sent using the `convertAndSend` method

```
jmsTemplate.convertAndSend(destination, text);
```

Receiving Messages

- A message can be received from a destination using the `receive` method of the Spring JMS template
- The method blocks until the message becomes available or the configured timeout is exceeded

```
Message message = jmsTemplate.receive(destination);
double version = message.getDoubleVersion();
String text = ((TextMessage) message).getText();
...
```

- If the message body can be converted by a configured message converter, a message can be received using the `receiveAndConvert` method

```
String text = (String) jmsTemplate.receiveAndConvert(destination);
```

Asynchronously Receiving Messages

- A message can asynchronously be received from a destination by annotating a callback method with `@JmsListener`

```
@JmsListener(destination = "...")
public void onMessage(Message message) {
    String text = ((TextMessage) message).getText();
    ...
}
```

- If the message body can be converted by a configured message converter, the type of the body can be used as parameter type

```
@JmsListener(destination = "...")
public void onMessage(String text) {
    ...
}
```

- The annotation’s concurrency element can be used to define the number of concurrent consumers (default is 1)

Selectively Receiving Messages

- Messages can selectively be received from a destination by providing a selector to the `receive` method or by using the selector attribute of the `@JmsListener` annotation
- A selector is an SQL-like conditional expression based on the properties of the message

```
Message message = jmsTemplate.receive(destination, "version > 2");
String text = ((TextMessage) message).getText();
...
```

```
@JmsListener(destination = "...", selector = "version > 2")
public void onMessage(Message message) {
    String text = ((TextMessage) message).getText();
    ...
}
```

Message Converter

- By default, the `SimpleMessageConverter` is able to handle messages of type `TextMessage`, `BytesMessage`, `MapMessage` and `ObjectMessage`
- A custom message converter can be provided by implementing the `MessageConverter` interface

```
@Component
public class CustomMessageConverter implements MessageConverter {

    public Object fromMessage(Message message)
        throws JmsException, MessageConversionException {
        ...
    }
    public Message toMessage(Object object, Session session)
        throws JmsException, MessageConversionException {
        ...
    }
}
```

JSON Message Converter

- Spring JMS provides the `JacksonJsonMessageConverter` that can convert messages to and from JSON
- The converter must be configured using a type ID property and ID to class mappings to allow conversion of different message types

```
@Configuration
public class JmsConfig {
    @Bean
    public MessageConverter jacksonJsonMessageConverter() {
        JacksonJsonMessageConverter converter =
            new JacksonJsonMessageConverter();
        converter.setTargetType(MessageType.TEXT);
        converter.setTypeIdPropertyName("typeId");
        Map<String, Class> mappings = Map.of(...);
        converter.setTypeIdMappings(mappings);
        return converter;
    }
}
```

Sending Messages and Receiving Replies

- The method `sendAndReceive` can be used to send a request message to a destination and to receive a reply message
- A temporary queue is created and automatically set in the `JMSReplyTo` header of the request message

```
Message reply = jmsTemplate.sendAndReceive(destination, session -> {
    Message message = session.createTextMessage(text);
    ...
    return message;
);
```

Sending Reply Messages

- Replies to request messages are sent to the temporary queue provided in the request's `JMSReplyTo` header
- Optionally, the `JMSCorrelationID` header of the reply message can be set to the request's message identifier

```
jmsTemplate.send(request.getJMSReplyTo(), session -> {
    TextMessage reply = session.createTextMessage(text);
    reply.setJMSCorrelationID(request.getJMSMessageID());
    return reply;
});
```