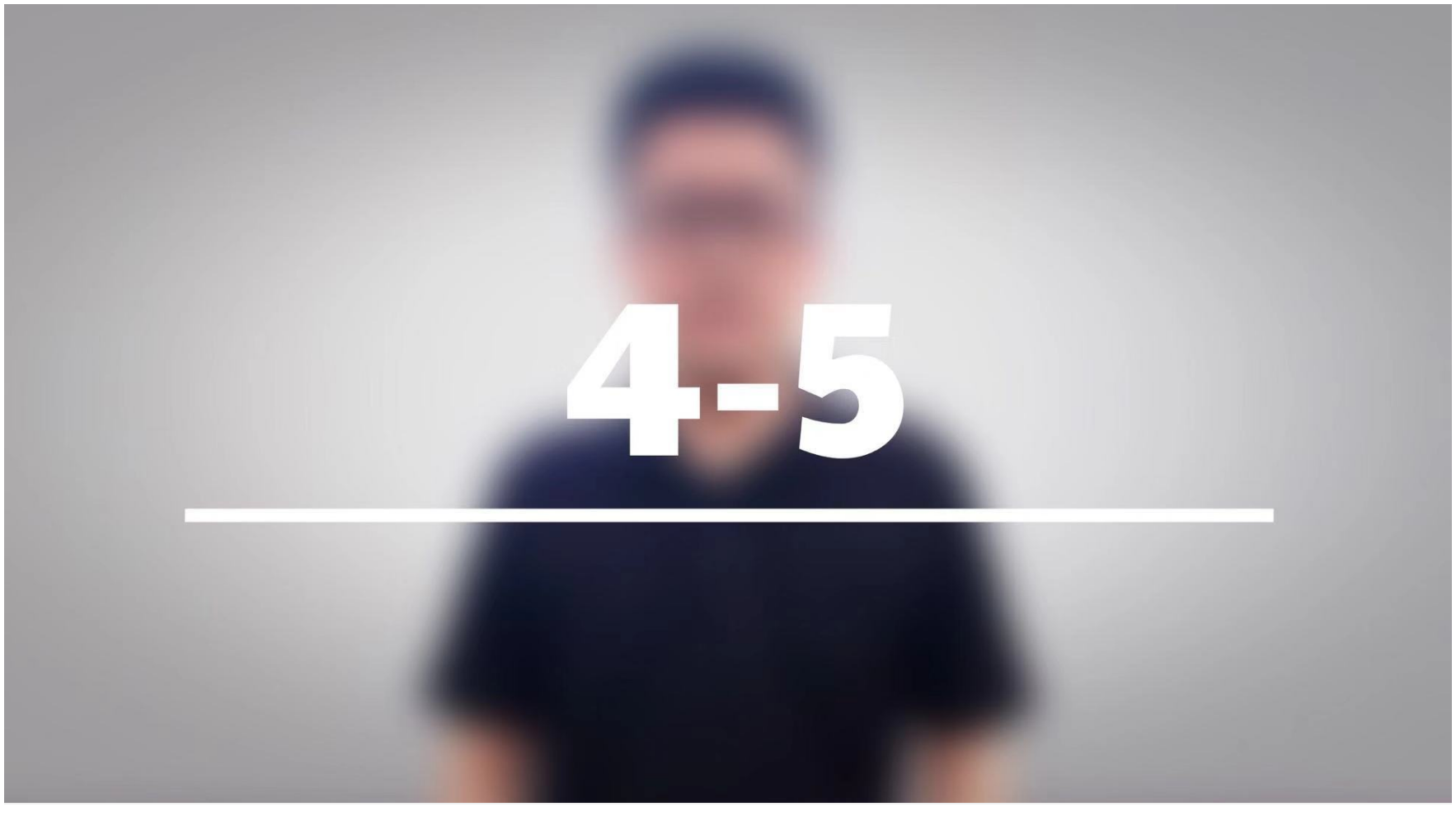

第 4-5 课：使用 Spring Boot 操作 ActiveMQ

消息队列中间件是分布式系统中重要的组件，主要解决应用耦合、异步消息、流量削锋等问题，实现高性能、高可用、可伸缩和最终一致性架构，是大型分布式系统不可缺少的中间件。

目前在生产环境中使用较多的消息队列有 ActiveMQ、RabbitMQ、ZeroMQ、Kaf - ka、MetaMQ、RocketMQ 等。

特性

- 异步性：将耗时的同步操作通过以发送消息的方式进行了异步化处理，减少了同步等待的时间。
- 松耦合：消息队列减少了服务之间的耦合性，不同的服务可以通过消息队列进行通信，而不用关心彼此的实现细节，只要定义好消息的格式就行。
- 分布式：通过对消费者的横向扩展，降低了消息队列阻塞的风险，以及单个消费者产生单点故障的可能性（当然消息队列本身也可以做成分布式集群）。
- 可靠性：消息队列一般会把接收到的消息存储到本地硬盘上（当消息被处理完之

后，存储信息根据不同的消息队列实现，有可能将其删除），这样即使应用挂掉或者消息队列本身挂掉，消息也能够重新加载。

JMS 规范

JMS 即 Java 消息服务（Java Message Service）应用程序接口，是一个 Java 平台中关于面向消息中间件（MOM）的 API，用于在两个应用程序之间，或分布式系统中发送消息，进行异步通信。Java 消息服务是一个与具体平台无关的 API，绝大多数 MOM 提供商都对 JMS 提供支持。

JMS 的消息机制有 2 种模型，一种是 Point to Point，表现为队列的形式，发送的消息，只能被一个接收者取走；另一种是 Topic，可以被多个订阅者订阅，类似于群发。

ActiveMQ 是 JMS 的一个实现。

ActiveMQ 介绍

ActiveMQ 是 Apache 软件基金下的一个开源软件，它遵循 JMS1.1 规范（Java Message Service），是消息驱动中间件软件（MOM）。它为企业消息传递提供高可用、出色性能、可扩展、稳定和安全保障。ActiveMQ 使用 Apache 许可协议，因此，任何人都可以使用和修改它而不必反馈任何改变。

ActiveMQ 的目标是在尽可能多的平台和语言上提供一个标准的，消息驱动的应用集成。ActiveMQ 实现 JMS 规范并在此之上提供大量额外的特性。ActiveMQ 支持队列和订阅两种模式的消息发送。

Spring Boot 提供了 ActiveMQ 组件 spring-boot-starter-activemq，用来支持 ActiveMQ 在 Spring Boot 体系内使用，下面我们来详细了解如何使用。

添加依赖

主要添加组件：spring-boot-starter-activemq。

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-activemq</artifactId>
</dependency>
```

配置文件

在 application.properties 中添加配置。

```
# 基于内存的 ActiveMQ
spring.activemq.in-memory=true
# 不适应连接池
spring.activemq.pool.enabled=false

# 独立安装的 ActiveMQ
#spring.activemq.broker-url=tcp://192.168.0.1:61616
#spring.activemq.user=admin
#spring.activemq.password=admin
```

在使用 ActiveMQ 时有两种使用方式，一种是使用独立安装的 ActiveMQ，在生产环境推荐使用这种；另一种是使用基于内存 ActiveMQ ， 在调试阶段建议使用这种方式。

队列（Queue）

队列发送的消息，只能被一个消费者接收。

创建队列

```
@Configuration
public class MqConfig {
    @Bean
    public Queue queue() {
        return new ActiveMQQueue("neo.queue");
    }
}
```

使用 @Configuration 注解在项目启动时，定义了一个队列 queue 命名为：neo.queue。

消息生产者

创建一个消息的生产者：

```
@Component
public class Producer{
    @Autowired
    private JmsMessagingTemplate jmsMessagingTemplate;
    @Autowired
    private Queue queue;
    public void sendQueue(String msg) {
        System.out.println("send queue msg :"+msg);
        this.jmsMessagingTemplate.convertAndSend(this.queue, msg);
    }
}
```

JmsMessagingTemplate 是 Spring 提供发送消息的工具类，使用 JmsMessaging Template 和创建好的 queue 对消息进行发送。

消息消费者

```
@Component
public class Consumer {

    @JmsListener(destination = "neo.queue")
    public void receiveQueue(String text) {
        System.out.println("Consumer queue msg : "+text);
    }
}
```

使用注解 @JmsListener(destination = "neo.queue")，表示此方法监控了名为 neo.queue 的队列。当队列 neo.queue 中有消息发送时会触发此方法的执行，text 为消息内容。

测试

创建 SampleActiveMqTests 测试类，注入创建好的消息生产者。

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class SampleActiveMqTests {
    @Autowired
    private Producer producer;
    @Rule
    public OutputCapture outputCapture = new OutputCapture();
}
```

OutputCapture 是 Spring Boot 提供的一个测试类，它能捕获 System.out 和 System.err 的输出，我们可以利用这个特性来判断程序中的输出是否执行。

```
@Test
public void sendSimpleQueueMessage() throws InterruptedException {
    this.producer.sendQueue("Test queue message");
    Thread.sleep(1000L);
    assertThat(this.outputCapture.toString().contains("Test queue")
).isTrue();
}
```

创建测试方式，使用 producer 发送消息，为了保证容器可以接收到消息，让测试方法等待 1 秒，最后使用 outputCapture 判断是否执行成功。

测试多消费者

上面的案例只是一个生产者一个消费者，我们在模拟一个生产者和多个消费者队列的执行情况。我们复制上面的消费者 Consumer 重新命名为 Consumer2，并且将输出内容加上 2 的关键字，如下：

```
@Component
public class Consumer2 {
    @JmsListener(destination = "neo.queue")
    public void receiveQueue(String text) {
        System.out.println("Consumer2 queue msg : "+text);
    }
}
```

在刚才的测试类中添加一个 send100QueueMessage() 方法，模式发送 100 条消息时，两个消费者是如何消费消息的。

```
@Test
public void send100QueueMessage() throws InterruptedException {
    for (int i=0;i<100;i++){
        this.producer.sendQueue("Test queue message"+i);
    }
    Thread.sleep(1000L);
}
```

控制台输出结果:

```
Consumer queue msg : Test queue message0
Consumer2 queue msg : Test queue message1
Consumer queue msg : Test queue message2
Consumer2 queue msg : Test queue message3
...
```

根据控制台输出的消息可以看出，当有多个消费者监听一个队列时，消费者会自动均衡负载的接收消息，并且每个消息只能有一个消费者所接收。

> 注意：控制台输出 javax.jms.JMSException: peer (vm://localhost#1) stopped. 报错信息可以忽略，这是 Info 级别的错误，是 ActiveMQ 的一个 bug。

广播（Topic）

广播发送的消息，可以被多个消费者接收。

创建 Topic

```
@Configuration
public class MqConfig {
    @Bean
    public Topic topic() {
        return new ActiveMQTopic("neo.topic");
    }
}
```

使用 @Configuration 注解在项目启动时，定义了一个广播 Topic 命名为：neo.topic。

消息生产者

创建一个消息的生产者:

```
@Component
public class Producer{
    @Autowired
    private JmsMessagingTemplate jmsMessagingTemplate;
    @Autowired
    private Topic topic;
    public void sendTopic(String msg) {
        System.out.println("send topic msg :"+msg);
        this.jmsMessagingTemplate.convertAndSend(this.topic, msg);
    }
}
```

和上面的生产者对比只是 convertAndSend() 方法传入的第一个参数变成了 Topic。

消息消费者

```
@Component
public class Consumer {

    @JmsListener(destination = "neo.topic")
    public void receiveTopic(String text) {
        System.out.println("Consumer topic msg : "+text);
    }
}
```

消费者也没有变化，只是监听的名改为上面的 neo.topic，因为模拟多个消费者，复制一份 Consumer 命名为 Consumer2，代码相同在输出中标明来自 Consumer2。

测试

创建 SampleActiveMqTests 测试类，注入创建好的消息生产者。

```
@Test
public void sendSimpleTopicMessage() throws InterruptedException {
    this.producer.sendTopic("Test Topic message");
    Thread.sleep(1000L);
}
```

测试方法执行成功后，会看到控制台输出信息，如下：

```
send topic msg :Test Topic message
Consumer topic msg : Test Topic message
Consumer2 topic msg : Test Topic message
```

可以看出两个消费者都收到了发送的消息，从而验证广播（Topic）是一个发送者多个消费者的模式。

同时支持队列（Queue）和广播（Topic）

Spring Boot 集成 ActiveMQ 的项目默认只支持队列或者广播中的一种，通过配置项 spring.jms.pub-sub-domain 的值来控制，true 为广播模式，false 为队列模式，默认情况下支持队列模式。

如果需要在同一项目中既支持队列模式也支持广播模式，可以通过 DefaultJmsLis - tenerContainerFactory 创建自定义的 JmsListenerContainerFactory 实例，之后在 @JmsListener 注解中通过 containerFactory 属性引用它。

分别创建两个自定义的 JmsListenerContainerFactory 实例，通过 pubSubDomain 来控制是支持队列模式还是广播模式。

```
@Configuration
@EnableJms
public class ActiveMQConfig {

    @Bean("queueListenerFactory")
    public JmsListenerContainerFactory<?> queueListenerFacto-
ry(ConnectionFactory connectionFactory) {
        DefaultJmsListenerContainerFactory factory = new DefaultJm-
sListenerContainerFactory();
        factory.setConnectionFactory(connectionFactory);
        factory.setPubSubDomain(false);
        return factory;
    }

    @Bean("topicListenerFactory")
    public JmsListenerContainerFactory<?> topicListenerFacto-
ry(ConnectionFactory connectionFactory) {
        DefaultJmsListenerContainerFactory factory = new DefaultJm-
sListenerContainerFactory();
        factory.setConnectionFactory(connectionFactory);
        factory.setPubSubDomain(true);
        return factory;
    }
}
```

然后在消费者接收的方法中，指明使用 containerFactory 接收消息。

```
@Component
public class Consumer {

    @JmsListener(destination = "neo.queue", containerFactory = "que
ueListenerFactory")
    public void receiveQueue(String text) {
        System.out.println("Consumer queue msg : "+text);
    }

    @JmsListener(destination = "neo.topic", containerFactory = "top
icListenerFactory")
    public void receiveTopic(String text) {
        System.out.println("Consumer topic msg : "+text);
    }
}
```

改造完成之后，再次执行队列和广播的测试方法，就会发现项目同时支持了两种类型的消息收发。

总结

消息中间件广泛应用在大型互联网架构中，利用消息中间件队列和广播各自的特性可以支持很多业务，比如群发发送短信、给单个用户发送邮件等。ActiveMQ 是一款非常流行的消息中间件，它的特点是部署简单、使用方便，比较适合中小型团队。Spring Boot 提供了集成 ActiveMQ 对应的组件，在 Spring Boot 中使用 ActiveMQ 只需要添加相关注解即可。

> 点击这里下载源码 (https://github.com/ityouknow/spring-boot-leaning/tree/gitbook_column2.0)。

(/gitchat/column/5b86228ce15aa17d68b5b55a/topic/5c0f4bda459532457215397 3)(/gitch