Spring消息

1 异步消息

同步通讯机制限制：

- 同步通讯意味着等待。当客户端调用远程服务的方法是，它必须等待远程方法结束后才能继续执行。如果客户端与远程服务频繁通讯，或者远程服务响应很慢，就会对客户端应用的性能带来负面影响
- 客户端通过服务接口与远程服务相耦合。如果服务的接口发生变化，此服务与客户端都需要做响应的改变
- 客户端与远程服务的位置相耦合。客户端必须配置服务的网络位置，这样他辞职到如何与远程服务进行交互。如果网络拓扑进行调整，客户端也需要重新配置新的网络位置。
- 客户端与服务的可用性相耦合。如果远程服务不可用，客户端实际上也无法正常运行。

异步消息优点；

- **无需等待。**发送消息时，客户端不必等待消息处理，甚至是被投递。客户端只需要将消息发送给消息代理，就可以确信i奥西会被投递给相应的目的地
- **面向消息和解耦。**异步消息以数据为中心，客户端并没有与特定的方法签名绑定。让你和可以处理数据的队列或主题订阅者都可以处理由客户端发送的消息，而客户端不必了解远程服务的任何规范。
- **位置独立。**消息客户端不必知道谁会处理他们的消息，或者服务的位置在哪儿。客户端只需要了解通过哪个队列或主题来发送消息。因此，只要服务呢个够从队列或主题中获取消息即可。
- **确保投递。**发送异步消息，客户端完全可以详细消息会被投递。即使在消息方式时，服务无法是哟共，消息也会被存储起来，知道服务重新可以是哟共为止。

2 使用JMS发送消息

Java消息服务(Java Message Service， JMS)是一个Java标准，定义了使用消息代理的通用API。

Spring通过基于模板的抽象为JMS功能提供了支持，JmsTemplate。使用JmsTemplate，能够非常容易地在消息生产方发送队列和主题消息，在消费消息的哪一方，也能够非常容易地接受这些消息。

Spring还提供了消息驱动POJO的理念：**一个简单的Java对象，它能够以异步的方式响应队列或主题上到达的消息**

2.1 在Spring中搭建消息代理

2.1.1 创建连接工厂

AcrtiveMQ--开源消息代理，使用JMS进行异步消息放传递

```
<amq:connectionFactory id="connectionFactory"
                       brokerURL="tcp://localhost:61616"/>
```

amq:connectionFactory元素是为ActiveMQ所准备的。如果使用不同的消息代理实例，他们不一定会提供Srping配置命名空间。此时需要使用bean

```
<bean id="connectionFactory"
      class="org.apache.activemq.spring.ActiveMQConnectionFactory"
      p:brokerURL="tcp://localhost:61616" />
```

2.1.2 声明ActiveMQ消息目的地

除了连接工厂外，还哦需要消息传递的目的地。目的地可以是一个队列，也可以是一个主题。不论是那个，都必须使用特定的消息代理实现类在Spring中配置目的地bean

声明一个ActiveMQ队列：

```
<bean id="spittleQueue" class="org.apache.activemq.command.ActiveMQQueue"
      c:_="spittle.alert.queue" />
```

声明一个ActiveMQ主题：

```
<bean id="spittleTopic" class="org.apache.activemq.command.ActiveMQTopic"
      c:_="spittle.alert.topic" />
```

与连接工厂类似，命名空间体哦国内了另一种共方式来声明队列和主题

```
<amq:queue id="spittleQueue" physicalName="spittle.alert.queue"/>

<amq:topic id="spittleTopic" physicalName="spittle.alert.topic"/>
```

2.2 使用Spring的JMS模板

JMS为Java开发者提供了与消息代理进行交互来发送和接受消息的标准，几乎每个消息代理实现都支持JMS。

2.2.1 处理失控的代码

只有及哈根代码是员工来发送消息，剩下的仅仅是为了发送消息而进行的设置(主要是异常处理)

2.2.2 使用JMS模板

针对如何消除冗长和重复的JMS代码，Spring给出的解决方案是JmsTemplate。

- JmsTemplate可以创建连接、获得绘画以及发送和接收消息。
- JmsTemplate可以处理所有抛出的笨拙的JmsTemplate异常。如果在使用是抛出JMSException，JmsTemplate将捕获该异常，然后派出一个非检查型异常，该异常是Spring自带的JmsException异常的子类

表17.1 Spring的JmsTemplate会捕获标准的JMSException异常，再以Spring的非检查型异常JmsException子类重新抛出

Spring（org.springframework.jms.*）	标准的JMS（javax.jms.*）
DestinationResolutionException	Spring特有的——当Spring无法解析目的地名称时抛出
IllegalStateException	IllegalStateException
InvalidClientIDException	InvalidClientIDException
InvalidDestinationException	InvalidSelectorException
InvalidSelectorException	InvalidSelectorException
JmsSecurityException	JmsSecurityException
ListenerExecutionFailedException	Spring特有的——当监听器方法执行失败时抛出
MessageConversionException	Spring特有的——当消息转换失败时抛出
MessageEOFException	MessageEOFException
MessageFormatException	MessageFormatException
MessageNotReadableException	MessageNotReadableException
MessageNotWriteableException	MessageNotWriteableException
ResourceAllocationException	ResourceAllocationException
SynchedLocalTransactionFailedException	Spring特有的——当同步的本地事务不能完成时抛出
TransactionInprogressException	TransactionInprogressException
TransactionRolledBackException	TransactionRolledBackException
UncategorizedJmsException	Spring特有的——当没有其他异常适用时抛出

使用JmsTemplate

```
<bean id="jmsTemplate"
      class="org.springframework.jms.core.JmsTemplate"
      c:_-ref="connectionFactory"/>
```

因为JmsTemplate需要知道如何连接到消息代理，所以需要为connectionFactory属性设置实现了JMS的ConnectionFactory接口的bean引用。

2.2.3 发送消息

```
public interface AlertService {

    void sendSpittleAlert(Spittle spittle);

    Spittle retrieveSpittleAlert();
}
```

```
package spittr.alerts;

import org.springframework.jms.core.JmsOperations;

import spittr.domain.Spittle;

public class AlertServiceImpl implements AlertService {

    private JmsOperations jmsOperations;

    public AlertServiceImpl(JmsOperations jmsOperations) {
        this.jmsOperations = jmsOperations;
    }

  public void sendSpittleAlert(final Spittle spittle) {
    jmsOperations.send(
      "spittle.alert.queue",//指定目的地
      new MessageCreator() {
        public Message createMessage(Session session)
                        throws JMSException {
          return session.createObjectMessage(spittle);//创建消息
        }
      }
    );
  }
}
```

JmsTemplate 会处理连接或会话管理。

2.2.4 设置默认目的地

```
<bean id="jmsTemplate"
      class="org.springframework.jms.core.JmsTemplate"
      c:_-ref="connectionFactory"
      p:defaultDestinationName="spittle.alert.queue"/>
```

如果一斤存在该名称的队列或煮汤，就会使用已有的。如果为存在，则会创建一个新的目的地(通常是队列)。如果要制定创建的目的地类型的化，可以将之前创建的队列或目的地的bean装配起来

```
<bean id="jmsTemplate"
      class="org.springframework.jms.core.JmsTemplate"
      c:_-ref="connectionFactory"
      p:defaultDestination-ref="spittleTopic"/>
```

现在调用JmsTemplate的send()方法，可以去除第一个参数

```
public void sendSpittleAlert(final Spittle spittle) {
    jmsOperations.send(
        new MessageCreator() {
            public Message createMessage(Session session)throws JMSException {
                return session.createObjectMessage(spittle);
            }
        }
    );
}
```

2.2.5 在发送时对消息进行转化

JmsTemplate提供了convertAndSend()方法。convertAndSend不需要MessageCreator作为参数。因为convertAndSend会是哟共内置的消息转换器创建消息

```
public void sendSpittleAlert(Spittle spittle) {
    jmsOperations.convertAndSend(spittle);
}
```

表17.2 Spring为通用的转换任务提供了多个消息转换器
（所有的消息转换器都位于org.springframework.jms.support.converter包中）

消息转换器	功　能
MappingJacksonMessageConverter	使用Jackson JSON库实现消息与JSON格式之间的相互转换
MappingJackson2MessageConverter	使用Jackson 2 JSON库实现消息与JSON格式之间的相互转换
MarshallingMessageConverter	使用JAXB库实现消息与XML格式之间的相互转换
SimpleMessageConverter	实现String与TextMessage之间的相互转换，字节数组与BytesMessage之间的相互转换，Map与MapMessage之间的相互转换以及Serializable对象与ObjectMessage之间的相互转换

默认情况，JmsTemplate在convertAndSend中使用SimpleMessageConverter，但是通过消息转换器声明为bean并将其注入到JmsTemplate的messageConverter属性中。

```
<bean id="messageConverter"

 class="org.springframework.jms.support.converter.MappingJacksonMessageConverter
"/>

<bean id="jmsTemplate"
      class="org.springframework.jms.core.JmsTemplate"
      c:_-ref="connectionFactory"
      p:defaultDestinationName="spittle.alert.queue"
      p:messageConverter-ref="messageConverter" />
```

2.2.6 接受消息

```
public Spittle getSpittleAlert() {
    try {
        ObjectMessage message = (ObjectMessage) jmsOperations.receive();
        return (Spittle) message.getObject();
    } catch (JMSException e) {
        throw JmsUtils.convertJmsAccessException(e);
    }
}
```

JmsTemplate当调用的recceive方法时，JmsTemplate会尝试从消息代理中获取一个消息。如果没有消息，recceive会一直等待，知道获得消息为止。

另外一种方式，可以

```
public Spittle retrieveSpittleAlert() {
    return (Spittle) jmsOperations.receiveAndConvert();
}
```

receive 和 receiveAndConvert都是同步的。如果没有消息，方法一直阻塞

2.3 创建消息驱动的POJO

解决接受阻塞。消息驱动bean (message-driven bean, MDB)是可以异步处理消息的EJB。

消息驱动POJO (MDP)用来异步接收消息

2.3.1 创建消息监听器

```
public class SpittleAlertHandler {

    public void handleSpittleAlert(Spittle spittle) {
        System.out.println(spittle.getMessage());
    }
}
```

是要给纯粹额POJO。他仍然可以想EJB那样处理消息，只不过他还需要一些spring的配置。

2.3.2 配置消息监听器

为POJO赋予消息接受能力的诀窍是在Spring中把它配置为消息监听器。

首先将POJO声明为bean

```
<bean id="spittleHandler" class="spittr.alerts.SpittleAlertHandler"/>
```

把这个bean声明为消息监听器

```
<jms:listener-container>
    <jms:listener destination="spittle.alert.queue"
                  ref="spittleHandler"
                  method="handleSpittleAlert"/>
</jms:listener-container>
```

在消息监听容器中包含了一个消息监听器。

消息监听容器：是一个特殊的bean，它可以监控JMS目的地并等待消息到达，一旦有消息到达，它取出消息，然后包消息传给任意一个对此消息甘心去的消息监听器。

如果ref属性所标示的bean实现了MessageListener，那就没必要在指定method属性，默认调用onMessage

2.4 使用基于消息的RPC

通过队列和主题在应用程序之间发送消息。

为了支持基于消息的RPC。**Spring提供了JmsInvokerServiceExporter，可以把bean代持股因为基于消息的服务；为客户端提供了JmsInvokerProxyFactoryBean来使用这些服务。**

2.4.1 导出基于JMS的服务

3 使用AMQP实现消息功能

AMQP不会直接将夏曦发布到队列中。AMQP在消息的生产者以及传递信息的队列之间引入了一种简介的机制：Exchange。Exchange会绑定到一个或多个队列上，它负责将信息路由到队列上。信息的消费者会从队列中提取数据并处理。

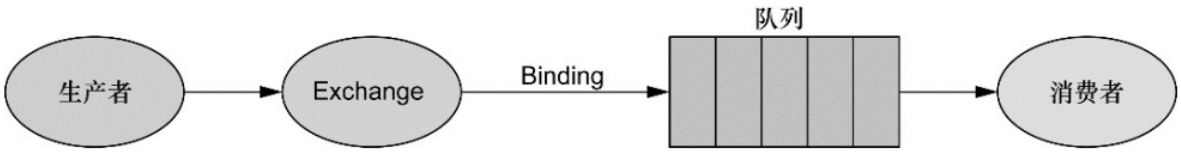

图17.8 在AMQP中，通过引入处理信息路由的Exchange，
消息的生产者 与消息队列之间实现了解耦

Exchange不是简单地将信息传递到队列中共，并不仅仅是一种穿透机制。AMQP定义了四种不同类型的Exchange，每一种都有不同的路由算法。

- **Direct：如果消息的routing key 与binding的routing key直接匹配的化，消息会路由到该队列上**
- **Topic 如果消息的routing key 与binding的routing key符合通配符匹配的化，消息会路由到该队列上**
- **Headers ： 如果消息参数表的头信息和值都与binding参数表中共相匹配，消息会路由到该队列上**
- **Fanout：不管消息的routing key和参数的头信息/值是什么，消息会路由到该队列上**

不再局限于JMS的点对点和发布-订阅的方式。

3.1配置Spring支持AMQP

RabbitMQ 实现了AMQP。Spring AMQP提供了RabbitMQ的连接工厂、模板以及Spring配置命名空间

3.1.1 连接工厂

```
<connection-factory id="connectionFactory"/>
```

3.1.2 声明队列、ExChange以及binding

元素	作用
	创建队列
	创建一个fanout类型的Exchange
	创建一个header类型的Exchange
	创建一个topic类型的Exchange
	创建一个dorect类型的Exchange
	元素定义一个或多个元素的集合。元素创建Exchange和队列之间的binging

这些元素要与元素一起使用。元素会创建一个RabbitMQ管理组件，它会自动创建上述这些元素所申明的队列、Exhcnage以及binging

```
<admin connection-factory="connectionFactory"/>

<queue id="spittleAlertQueue" name="spittle.alerts"/>
```

这样配置默认会有一个没有名称的directExchange，所有队列都会绑定到这个Exchnage，并且routing key与队列的名称相同。在这个配置中，可以将消息发送到这个没有名称的Exchange上，并将routing key设定为"spittle.alert.queue"，这样消息就会路由到这个队列中。实际上，重新创建了JMS的点对点模型。

```
<admin connection-factory="connectionFactory" />
<queue name="spittle.alert.queue.1" >
<queue name="spittle.alert.queue.2" >
<queue name="spittle.alert.queue.3" >
<fanout-exchange name="spittle.fanout">
 <bindings>
  <binding queue="spittle.alert.queue.1" />
  <binding queue="spittle.alert.queue.2" />
  <binding queue="spittle.alert.queue.3" />
 </bindings>
</fanout-exchange>
```

3.2 使用RabbitTemplate发送消息

```
<template id="rabbitTemplate"
          connection-factory="connectionFactory"
          routing-key="spittle.alerts"/>
```

```
public class AlertServiceImpl implements AlertService {

    private RabbitTemplate rabbit;

    @Autowired
    public AlertServiceImpl(RabbitTemplate rabbit) {
        this.rabbit = rabbit;
    }

    public void sendSpittleAlert(Spittle spittle) {
        rabbit.convertAndSend("spittle.alert.exchange",
```

```
                "spittle.alerts",
                spittle);
    }

}
```

如果在参数列表中共省略Exchange名称，或者同时省略Exhcnage名称和routing key的话，RabbitTemplate将会使用默认的Exchange名称和routing key。按照之前的配置莫仍的Exchange名称为空，默认的routing key也为空。**我们可以在**