Jackson2JsonMessageConverter与ContentTypeDelegatingMessageConverter详解

我们工作中各服务之间大多数数据都是以JSON类型的数据进行传输的，即生产者服务将JSON类型的数据传递到对应的队列， 而消费端处理器中接收到的数据类型也是JSON类型。

Jackson2JsonMessageConverter

消息转换器使用了RabbitMQ自带的Jackson2JsonMessageConverter转换器，但是没有指定消息的contentType类型。

消费端 :

|  |
| --- |
| @Configuration **public class** MQConfig {   @Bean  **public** ConnectionFactory connectionFactory(){  CachingConnectionFactory factory = **new** CachingConnectionFactory();  factory.setUri(**"amqp://zhihao.miao:123456@192.168.1.131:5672"**);  **return** factory;  }   @Bean  **public** RabbitAdmin rabbitAdmin(ConnectionFactory connectionFactory){  RabbitAdmin rabbitAdmin = **new** RabbitAdmin(connectionFactory);  **return** rabbitAdmin;  }   @Bean  **public** RabbitTemplate rabbitTemplate(ConnectionFactory connectionFactory){  RabbitTemplate rabbitTemplate = **new** RabbitTemplate(connectionFactory);  **return** rabbitTemplate;  }   @Bean  **public** SimpleMessageListenerContainer messageListenerContainer(ConnectionFactory connectionFactory){  SimpleMessageListenerContainer container = **new** SimpleMessageListenerContainer();  container.setConnectionFactory(connectionFactory);  container.setQueueNames(**"zhihao.miao.order"**);   MessageListenerAdapter adapter = **new** MessageListenerAdapter(**new** MessageHandler());  *//指定Json转换器* adapter.setMessageConverter(**new** Jackson2JsonMessageConverter());  *//设置处理器的消费消息的默认方法* adapter.setDefaultListenerMethod(**"onMessage"**);  container.setMessageListener(adapter);   **return** container;  } } |

处理器，定义了二个消息处理方法，参数不一样：

|  |
| --- |
| **public class** MessageHandler {   **public void** onMessage(**byte**[] message){  System.***out***.println(**"---------onMessage----byte-------------"**);  System.***out***.println(**new** String(message));  }    **public void** onMessage(String message){  System.***out***.println(**"---------onMessage---String-------------"**);  System.***out***.println(message);  } |

消费端应用启动类：

|  |
| --- |
| **import** org.springframework.context.annotation.AnnotationConfigApplicationContext; **import** org.springframework.context.annotation.ComponentScan; **import** java.util.concurrent.TimeUnit;  @ComponentScan **public class** Application {  **public static void** main(String[] args) **throws** Exception{  AnnotationConfigApplicationContext context = **new** AnnotationConfigApplicationContext(Application.**class**);  System.***out***.println(**"===start up======"**);  TimeUnit.SECONDS.sleep(60);  context.close();  } } |

生产端代码

|  |
| --- |
| **import** org.springframework.amqp.rabbit.connection.CachingConnectionFactory; **import** org.springframework.amqp.rabbit.connection.ConnectionFactory; **import** org.springframework.amqp.rabbit.core.RabbitAdmin; **import** org.springframework.amqp.rabbit.core.RabbitTemplate; **import** org.springframework.context.annotation.Bean; **import** org.springframework.context.annotation.Configuration;  @Configuration **public class** MQConfig {   @Bean  **public** ConnectionFactory connectionFactory(){  CachingConnectionFactory factory = **new** CachingConnectionFactory();  factory.setUri(**"amqp://zhihao.miao:123456@192.168.1.131:5672"**);  **return** factory;  }   @Bean  **public** RabbitAdmin rabbitAdmin(ConnectionFactory connectionFactory){  RabbitAdmin rabbitAdmin = **new** RabbitAdmin(connectionFactory);  **return** rabbitAdmin;  }   @Bean  **public** RabbitTemplate rabbitTemplate(ConnectionFactory connectionFactory){  RabbitTemplate rabbitTemplate = **new** RabbitTemplate(connectionFactory);  **return** rabbitTemplate;  } } |

消息的实体类型

|  |
| --- |
| **public class** Order {   **private** Integer **id**;  **private** Integer **userId**;  **private double amout**;  **private** String **time**;   **public** Integer getId() {  **return id**;  }   **public void** setId(Integer id) {  **this**.**id** = id;  }   **public** Integer getUserId() {  **return userId**;  }   **public void** setUserId(Integer userId) {  **this**.**userId** = userId;  }   **public double** getAmout() {  **return amout**;  }   **public void** setAmout(**double** amout) {  **this**.**amout** = amout;  }   **public** String getTime() {  **return time**;  }   **public void** setTime(String time) {  **this**.**time** = time;  }   @Override  **public** String toString() {  **return "Order{"** +  **"id="** + **id** +  **", userId="** + **userId** +  **", amout="** + **amout** +  **", time='"** + **time** + **'\''** +  **'}'**;  } } |

应用启动类，生产端传递的消息类型是Order类型，并且转换成JSON类型发送到队列中

|  |
| --- |
| **import** com.fasterxml.jackson.databind.ObjectMapper;  **import** org.springframework.amqp.rabbit.core.RabbitTemplate;  **import** org.springframework.context.annotation.AnnotationConfigApplicationContext;  **import** org.springframework.context.annotation.ComponentScan; **import** java.time.LocalDateTime;  @ComponentScan **public class** Application {  **public static void** main(String[] args) **throws** Exception{  AnnotationConfigApplicationContext context = **new** AnnotationConfigApplicationContext(Application.**class**);   RabbitTemplate rabbitTemplate = context.getBean(RabbitTemplate.**class**);  System.***out***.println(rabbitTemplate);   Order order = **new** Order();  order.setId(1);  order.setUserId(1000);  order.setAmout(88d);  order.setTime(LocalDateTime.now().toString());   ObjectMapper mapper = **new** ObjectMapper();  String json = mapper.writeValueAsString(order);  System.***out***.println(json);   rabbitTemplate.convertAndSend(**""**,**"zhihao.miao.order"**,json);  context.close();  } } |

消费之后的控制台打印：

|  |
| --- |
|  |

我们发现消费端还是将其当作字节数组来消费，转换器还是将其转换成byte[] 。

改造:

此时是因为生产端没有指定contentType类型，生产者应用启动类重新指定了相应的contentType类型后，

|  |
| --- |
| **import** com.fasterxml.jackson.databind.ObjectMapper;  **import** org.springframework.amqp.rabbit.core.RabbitTemplate;  **import** org.springframework.context.annotation.AnnotationConfigApplicationContext;  **import** org.springframework.context.annotation.ComponentScan; **import** java.time.LocalDateTime;  @ComponentScan **public class** Application {  **public static void** main(String[] args) **throws** Exception{  AnnotationConfigApplicationContext context = **new** AnnotationConfigApplicationContext(Application.**class**);   RabbitTemplate rabbitTemplate = context.getBean(RabbitTemplate.**class**);  System.***out***.println(rabbitTemplate);   Order order = **new** Order();  order.setId(1);  order.setUserId(1000);  order.setAmout(88d);  order.setTime(LocalDateTime.now().toString());   ObjectMapper mapper = **new** ObjectMapper();  String json = mapper.writeValueAsString(order);  System.***out***.println(json);  MessageProperties **messageProperties** = **new** MessageProperties();  messageProperties.setContentType(**"application/json"**);  Message message = **new** Message(json.getBytes(),messageProperties);  rabbitTemplate.convertAndSend(**""**,**"zhihao.miao.order"**,message);  context.close();  } } |

此时消费端的Jackson2JsonMessageConverter类型转换器将其转换成Map类型，指定消费的方法参数类型是Map即可。

|  |
| --- |
| **public class** MessageHandler {   **public void** onMessage(**byte**[] message){  System.***out***.println(**"---------onMessage----byte-------------"**);  System.***out***.println(**new** String(message));  }   **public void** onMessage(String message){  System.***out***.println(**"---------onMessage---String-------------"**);  System.***out***.println(message);  }   **public void** onMessage(Map order){  System.***out***.println(**"---------onMessage---map-------------"**);  System.***out***.println(order.toString());  }  } |

此时消费端控制台打印，我们知道生产者传递JSON类型数据，消费者将其作为Map类型的数据进行处理：

|  |
| --- |
|  |

再次改造:

如果消费端发送多条消息，发送List的json格式，那么在消费端也要使用参数是List的方法来消费，生产者启动应用类。

|  |
| --- |
| **import** com.fasterxml.jackson.databind.ObjectMapper;  **import** org.springframework.amqp.core.Message;  **import** org.springframework.amqp.core.MessageProperties;  **import** org.springframework.amqp.rabbit.core.RabbitTemplate;  **import** org.springframework.context.annotation.AnnotationConfigApplicationContext;  **import** org.springframework.context.annotation.ComponentScan;   **import** java.time.LocalDateTime;  **import** java.util.ArrayList;  **import** java.util.List;  @ComponentScan **public class** Application {  **public static void** main(String[] args) **throws** Exception{  AnnotationConfigApplicationContext context = **new** AnnotationConfigApplicationContext(Application.**class**);   RabbitTemplate rabbitTemplate = context.getBean(RabbitTemplate.**class**);  System.***out***.println(rabbitTemplate);   Order order = **new** Order();  order.setId(1);  order.setUserId(1000);  order.setAmout(88d);  order.setTime(LocalDateTime.now().toString());   ObjectMapper mapper = **new** ObjectMapper();  String json = mapper.writeValueAsString(order);  System.***out***.println(json);   MessageProperties messageProperties = **new** MessageProperties();  messageProperties.setContentType(**"application/json"**);  Message message = **new** Message(json.getBytes(),messageProperties);   rabbitTemplate.send(**""**,**"zhihao.miao.order"**,message);   Order order2 = **new** Order();  order2.setId(2);  order2.setUserId(2000);  order2.setAmout(99d);  order2.setTime(LocalDateTime.now().toString());   List<Order> orderList = **new** ArrayList<>();  orderList.add(order);  orderList.add(order2);   String jsonlist = mapper.writeValueAsString(orderList);  Message message2 = **new** Message(jsonlist.getBytes(),messageProperties);  rabbitTemplate.send(**""**,**"zhihao.miao.order"**,message2);   context.close();  } } |

消费端的Handler：

|  |
| --- |
| **public class** MessageHandler {   **public void** onMessage(**byte**[] message){  System.***out***.println(**"---------onMessage----byte-------------"**);  System.***out***.println(**new** String(message));  }    **public void** onMessage(String message){  System.***out***.println(**"---------onMessage---String-------------"**);  System.***out***.println(message);  }    **public void** onMessage(Map order){  System.***out***.println(**"---------onMessage---map-------------"**);  System.***out***.println(order.toString());  }   **public void** onMessage(List orders){  System.***out***.println(**"---------onMessage---List-------------"**);  System.***out***.println(orders.toString());  }  } |

消费者控制台打印,此时发现消费端将消息转换成List类型的消息：

|  |
| --- |
|  |

总结:

1、使用Jackson2JsonMessageConverter处理器，客户端发送JSON类型数据，但是没有指定消息的contentType类型，那么Jackson2JsonMessageConverter就会将消息转换成byte[]类型的消息进行消费。

2、如果指定了contentType为application/json，那么消费端就会将消息转换成Map类型的消息进行消费。

3、如果指定了contentType为application/json，并且生产端是List类型的JSON格式，那么消费端就会将消息转换成List类型的消息进行消费。

Jackson2JsonMessageConverter类的源码分析：

|  |
| --- |
| @Override **public** Object fromMessage(Message message)  **throws** MessageConversionException {  Object content = **null**;  MessageProperties properties = message.getMessageProperties();  **if** (properties != **null**) {  String contentType = properties.getContentType();  *//contentType中包含有json的都是用指定的格式来转换消息* **if** (contentType != **null** && contentType.contains(**"json"**)) {  String encoding = properties.getContentEncoding();  **if** (encoding == **null**) {  encoding = getDefaultCharset();  }  **try** {   **if** (getClassMapper() == **null**) {  JavaType targetJavaType = getJavaTypeMapper()  .toJavaType(message.getMessageProperties());  content = convertBytesToObject(message.getBody(),  encoding, targetJavaType);  }  **else** {  Class<?> targetClass = getClassMapper().toClass(  message.getMessageProperties());  content = convertBytesToObject(message.getBody(),  encoding, targetClass);  }  }  **catch** (IOException e) {  **throw new** MessageConversionException(  **"Failed to convert Message content"**, e);  }  }  **else** {  **if** (log.isWarnEnabled()) {  log.warn(**"Could not convert incoming message with content-type ["** + contentType + **"]"**);  }  }  }  *//其余的使用* **if** (content == **null**) {  content = message.getBody();  }  **return** content; } |

结论：

Jackson2JsonMessageConverter如果接收到的消息属性里面没有content\_type属性，或者content\_type值不包含json，则转换后的结果是byte[] 。

Jackson2JsonMessageConverter详解续

上面我们提到的是将实体类型转换成Map或者List类型，这样转换没有多大意义，我们需要消费者将生产者的消息对象格式转换成对应的消息格式，而不是Map或者List对象，解决方案，看代码：

生成端代码：

|  |
| --- |
| **import** com.fasterxml.jackson.databind.ObjectMapper;  **import** org.springframework.amqp.core.Message;  **import** org.springframework.amqp.core.MessageProperties;  **import** org.springframework.amqp.rabbit.core.RabbitTemplate;  **import** org.springframework.context.annotation.AnnotationConfigApplicationContext;  **import** org.springframework.context.annotation.ComponentScan;  **import** java.time.LocalDateTime;  */\*\*  \* 生产者在发送json数据的时候，需要指定这个json是哪个对象，否则消费者收到消息之后，不知道要转换成哪个java对象  \*  \* 指定方法：  \* 在消息header中，增加一个\_TypeId\_,value就是具体的java对象（全类名）,一定是消费者所在系统的java对象全称  \*/*  @ComponentScan **public class** Application {   **public static void** sendOrder( RabbitTemplate rabbitTemplate) **throws** Exception{  Order order = **new** Order();  order.setId(1);  order.setUserId(1000);  order.setAmout(88d);  order.setTime(LocalDateTime.now().toString());   ObjectMapper mapper = **new** ObjectMapper();  String json = mapper.writeValueAsString(order);  System.***out***.println(json);   MessageProperties messageProperties = **new** MessageProperties();  messageProperties.setContentType(**"application/json"**);  *//指定的\_\_TypeId\_\_属性值必须是消费端的Order的全类名，如果不匹配则会报错。* messageProperties.getHeaders().put(**"\_\_TypeId\_\_"**,**"com.zhihao.miao.test.day10.Sender.Order"**);  Message message = **new** Message(json.getBytes(),messageProperties);   rabbitTemplate.send(**""**,**"zhihao.miao.order"**,message);  }   **public static void** main(String[] args) **throws** Exception{  AnnotationConfigApplicationContext context = **new** AnnotationConfigApplicationContext(Application.**class**);  RabbitTemplate rabbitTemplate = context.getBean(RabbitTemplate.**class**);  System.***out***.println(rabbitTemplate);  *sendOrder*(rabbitTemplate);  context.close();  } } |

消费端的Handler改造：

|  |
| --- |
| **import** java.util.List;  **import** java.util.Map;  **import** com.zhihao.miao.test.day10.Sender.Order;  **public class** MessageHandler {   **public void** onMessage(**byte**[] message){  System.***out***.println(**"---------onMessage----byte-------------"**);  System.***out***.println(**new** String(message));  }    **public void** onMessage(String message){  System.***out***.println(**"---------onMessage---String-------------"**);  System.***out***.println(message);  }    **public void** onMessage(Map order){  System.***out***.println(**"---------onMessage---map-------------"**);  System.***out***.println(order.toString());  }   **public void** onMessage(Order order){  System.***out***.println(**"---------onMessage---Order-------------"**);  System.***out***.println(order);  }   **public void** onMessage(List orders){  System.***out***.println(**"---------onMessage---List-------------"**);  System.***out***.println(orders.toString());  }  } |

测试之后发现消费端调用的是onMessage(Order order)这个方法，消费端控制台打印：

|  |
| --- |
|  |

总结：

生产者在发送json数据的时候，需要指定这个json是哪个对象，否则消费者收到消息之后，不知道要转换成哪个java对象

指定方法：

在消息header中，增加一个TypeId,value就是具体的java对象（全类名）,一定是消费者所在系统的java对象全称。

优化:

我们发现生产者和消费者的耦合度太高，生产者需要知道消费者相应对应的全类名，如何去改造呢？

在消费端配置映射：

|  |
| --- |
| @Bean **public** SimpleMessageListenerContainer messageListenerContainer(ConnectionFactory connectionFactory){  SimpleMessageListenerContainer container = **new** SimpleMessageListenerContainer();  container.setConnectionFactory(connectionFactory);  container.setQueueNames(**"zhihao.miao.order"**);   MessageListenerAdapter adapter = **new** MessageListenerAdapter(**new** MessageHandler());  *//指定Json转换器* Jackson2JsonMessageConverter jackson2JsonMessageConverter =**new** Jackson2JsonMessageConverter();   *//消费端配置映射* Map<String, Class<?>> idClassMapping = **new** HashMap<>();  idClassMapping.put(**"order"**,Order.**class**);  idClassMapping.put(**"user"**,User.**class**);   DefaultJackson2JavaTypeMapper jackson2JavaTypeMapper = **new** DefaultJackson2JavaTypeMapper();  jackson2JavaTypeMapper.setIdClassMapping(idClassMapping);   System.***out***.println(**"在jackson2JsonMessageConverter转换器中指定映射配置"**);  jackson2JsonMessageConverter.setJavaTypeMapper(jackson2JavaTypeMapper);  adapter.setMessageConverter(jackson2JsonMessageConverter);   *//设置处理器的消费消息的默认方法* adapter.setDefaultListenerMethod(**"onMessage"**);  container.setMessageListener(adapter);   **return** container; } |

消费者处理器Handler中增加入参数是User的方法：

|  |
| --- |
| **import** java.util.List;  **import** java.util.Map;  **import** com.zhihao.miao.test.day10.Sender.Order;  **import** com.zhihao.miao.test.day10.Sender.User;  **public class** MessageHandler {   **public void** onMessage(**byte**[] message){  System.***out***.println(**"---------onMessage----byte-------------"**);  System.***out***.println(**new** String(message));  }    **public void** onMessage(String message){  System.***out***.println(**"---------onMessage---String-------------"**);  System.***out***.println(message);  }    **public void** onMessage(Map order){  System.***out***.println(**"---------onMessage---map-------------"**);  System.***out***.println(order.toString());  }   **public void** onMessage(Order order){  System.***out***.println(**"---------onMessage---Order-------------"**);  System.***out***.println(order);  }   **public void** onMessage(User user){  System.***out***.println(**"---------onMessage---user-------------"**);  System.***out***.println(user.toString());  }   **public void** onMessage(List orders){  System.***out***.println(**"---------onMessage---List-------------"**);  System.***out***.println(orders.toString());  }  } |

然后在生产端就可以指定对应的key，而不需要再去指定全类名了，

|  |
| --- |
| **import** com.fasterxml.jackson.databind.ObjectMapper;  **import** org.springframework.amqp.core.Message;  **import** org.springframework.amqp.core.MessageProperties;  **import** org.springframework.amqp.rabbit.core.RabbitTemplate;  **import** org.springframework.context.annotation.AnnotationConfigApplicationContext;  **import** org.springframework.context.annotation.ComponentScan; **import** java.time.LocalDateTime;  @ComponentScan **public class** Application {   **public static void** sendOrder( RabbitTemplate rabbitTemplate) **throws** Exception{  Order order = **new** Order();  order.setId(1);  order.setUserId(1000);  order.setAmout(88d);  order.setTime(LocalDateTime.now().toString());  System.***out***.println(order);   ObjectMapper mapper = **new** ObjectMapper();  String json = mapper.writeValueAsString(order);  System.***out***.println(json);   MessageProperties messageProperties = **new** MessageProperties();  messageProperties.setContentType(**"application/json"**);  messageProperties.getHeaders().put(**"\_\_TypeId\_\_"**,**"order"**);  Message message = **new** Message(json.getBytes(),messageProperties);   rabbitTemplate.send(**""**,**"zhihao.miao.order"**,message);  }   **public static void** sendUser( RabbitTemplate rabbitTemplate) **throws** Exception{  User user = **new** User();  user.setUserId(1000);  user.setAge(50);  user.setUsername(**"zhihao.miao"**);  user.setPassword(**"123343"**);  System.***out***.println(user);   ObjectMapper mapper = **new** ObjectMapper();  String json = mapper.writeValueAsString(user);  System.***out***.println(json);   MessageProperties messageProperties = **new** MessageProperties();  messageProperties.setContentType(**"application/json"**);  *//指定消费端配置的key值就行了* messageProperties.getHeaders().put(**"\_\_TypeId\_\_"**,**"user"**);  Message message = **new** Message(json.getBytes(),messageProperties);   rabbitTemplate.send(**""**,**"zhihao.miao.order"**,message);  }   **public static void** main(String[] args) **throws** Exception{  AnnotationConfigApplicationContext context = **new** AnnotationConfigApplicationContext(Application.**class**);   RabbitTemplate rabbitTemplate = context.getBean(RabbitTemplate.**class**);  System.***out***.println(rabbitTemplate);   *//sendOrder(rabbitTemplate);  sendUser*(rabbitTemplate);  context.close();  } } |

进行测试发现结果符合我们预期。

结论：

发送消息的时候,TypeId的值可以是java对象全称，也可以是映射的key

当消费者有配置映射key的时候，生产者既可以指定java对象全称，又可以是映射的key。如果消费者没有配置映射key，则只能指定java对象全称。

Jackson2JsonMessageConverter详解续

如果消息类型是List或者Map类型的时候。

生产端：

|  |
| --- |
| **public static void** sendOrderList(RabbitTemplate rabbitTemplate) **throws** Exception{  Order order = **new** Order();  order.setId(1);  order.setUserId(1000);  order.setAmout(88d);  order.setTime(LocalDateTime.now().toString());   Order order2 = **new** Order();  order2.setId(2);  order2.setUserId(2000);  order2.setAmout(99d);  order2.setTime(LocalDateTime.now().toString());   List<Order> orderList = Arrays.asList(order,order2);   ObjectMapper mapper = **new** ObjectMapper();  String json = mapper.writeValueAsString(orderList);   MessageProperties messageProperties = **new** MessageProperties();  messageProperties.setContentType(**"application/json"**);  messageProperties.getHeaders().put(**"\_\_TypeId\_\_"**,**"java.util.List"**);  messageProperties.getHeaders().put(**"\_\_ContentTypeId\_\_"**,**"order"**);    Message message = **new** Message(json.getBytes(),messageProperties);  rabbitTemplate.send(**""**,**"zhihao.miao.order"**,message); }   **public static void** sendOrderMap(RabbitTemplate rabbitTemplate) **throws** Exception{  Order order = **new** Order();  order.setId(1);  order.setUserId(1000);  order.setAmout(88d);  order.setTime(LocalDateTime.now().toString());   Order order2 = **new** Order();  order2.setId(2);  order2.setUserId(2000);  order2.setAmout(99d);  order2.setTime(LocalDateTime.now().toString());   Map<String,Object> orderMaps = **new** HashMap<>();  orderMaps.put(**"10"**,order);  orderMaps.put(**"20"**,order2);   ObjectMapper mapper = **new** ObjectMapper();  String json = mapper.writeValueAsString(orderMaps);   MessageProperties messageProperties = **new** MessageProperties();  messageProperties.setContentType(**"application/json"**);  messageProperties.getHeaders().put(**"\_\_TypeId\_\_"**,**"java.util.Map"**);  messageProperties.getHeaders().put(**"\_\_KeyTypeId\_\_"**,**"java.lang.String"**);  messageProperties.getHeaders().put(**"\_\_ContentTypeId\_\_"**,**"order"**);    Message message = **new** Message(json.getBytes(),messageProperties);  rabbitTemplate.send(**""**,**"zhihao.miao.order"**,message); } |

消费端：

|  |
| --- |
| **public void** onMessage(List<Order> orders){  System.***out***.println(**"---------onMessage---List<Order>-------------"**);  orders.stream().forEach(order -> System.***out***.println(order)); }  **public void** onMessage(Map<String,Object> orderMaps){  System.***out***.println(**"-------onMessage---Map<String,Object>------------"**);  orderMaps.keySet().forEach(key -> System.***out***.println(orderMaps.get(key))); } |

结论:

如果生产者发送的是list的json数据，则还需要增加一个\_\_ContentTypeId\_\_的header，用于指明List里面的具体对象。

如果生产者发送的是map的json数据，则需要指定\_\_KeyTypeId\_\_，\_\_ContentTypeId\_\_的header，用于指明map里面的key，value的具体对象。

ContentTypeDelegatingMessageConverter详解续

生产端：

|  |
| --- |
| **public class** Application {   **public static void** sendOrder( RabbitTemplate rabbitTemplate) **throws** Exception{  Order order = **new** Order();  order.setId(1);  order.setUserId(1000);  order.setAmout(88d);  order.setTime(LocalDateTime.now().toString());   ObjectMapper mapper = **new** ObjectMapper();  String json = mapper.writeValueAsString(order);  System.***out***.println(json);   MessageProperties messageProperties = **new** MessageProperties();  messageProperties.setContentType(**"application/json"**);  messageProperties.getHeaders().put(**"\_\_TypeId\_\_"**,**"order"**);  Message message = **new** Message(json.getBytes(),messageProperties);   rabbitTemplate.send(**""**,**"zhihao.miao.order"**,message);  }   **public static void** sendUser( RabbitTemplate rabbitTemplate) **throws** Exception{  User user = **new** User();  user.setUserId(1000);  user.setAge(50);  user.setUsername(**"zhihao.miao"**);  user.setPassword(**"123343"**);   ObjectMapper mapper = **new** ObjectMapper();  String json = mapper.writeValueAsString(user);  System.***out***.println(json);   MessageProperties messageProperties = **new** MessageProperties();  messageProperties.setContentType(**"application/json"**);  messageProperties.getHeaders().put(**"\_\_TypeId\_\_"**,**"user"**);  Message message = **new** Message(json.getBytes(),messageProperties);   rabbitTemplate.send(**""**,**"zhihao.miao.order"**,message);  }   **public static void** sendOrderList(RabbitTemplate rabbitTemplate) **throws** Exception{  Order order = **new** Order();  order.setId(1);  order.setUserId(1000);  order.setAmout(88d);  order.setTime(LocalDateTime.now().toString());   Order order2 = **new** Order();  order2.setId(2);  order2.setUserId(2000);  order2.setAmout(99d);  order2.setTime(LocalDateTime.now().toString());   List<Order> orderList = Arrays.asList(order,order2);   ObjectMapper mapper = **new** ObjectMapper();  String json = mapper.writeValueAsString(orderList);   MessageProperties messageProperties = **new** MessageProperties();  messageProperties.setContentType(**"application/json"**);  messageProperties.getHeaders().put(**"\_\_TypeId\_\_"**,**"java.util.List"**);  messageProperties.getHeaders().put(**"\_\_ContentTypeId\_\_"**,**"order"**);    Message message = **new** Message(json.getBytes(),messageProperties);  rabbitTemplate.send(**""**,**"zhihao.miao.order"**,message);  }    **public static void** sendOrderMap(RabbitTemplate rabbitTemplate) **throws** Exception{  Order order = **new** Order();  order.setId(1);  order.setUserId(1000);  order.setAmout(88d);  order.setTime(LocalDateTime.now().toString());   Order order2 = **new** Order();  order2.setId(2);  order2.setUserId(2000);  order2.setAmout(99d);  order2.setTime(LocalDateTime.now().toString());   Map<String,Object> orderMaps = **new** HashMap<>();  orderMaps.put(**"10"**,order);  orderMaps.put(**"20"**,order2);   ObjectMapper mapper = **new** ObjectMapper();  String json = mapper.writeValueAsString(orderMaps);   MessageProperties messageProperties = **new** MessageProperties();  messageProperties.setContentType(**"application/json"**);  messageProperties.getHeaders().put(**"\_\_TypeId\_\_"**,**"java.util.Map"**);  messageProperties.getHeaders().put(**"\_\_KeyTypeId\_\_"**,**"java.lang.String"**);  messageProperties.getHeaders().put(**"\_\_ContentTypeId\_\_"**,**"order"**);    Message message = **new** Message(json.getBytes(),messageProperties);  rabbitTemplate.send(**""**,**"zhihao.miao.order"**,message);  }    **public static void** sendJepg(RabbitTemplate rabbitTemplate) **throws** Exception{  **byte**[] body = Files.readAllBytes(Paths.get(**"/Users/naeshihiroshi/Desktop/file/file"**,**"aisi.jpeg"**));   MessageProperties messageProperties = **new** MessageProperties();  messageProperties.setContentType(**"image/jepg"**);   Message message = **new** Message(body,messageProperties);   rabbitTemplate.send(**""**,**"zhihao.miao.order"**,message);  }   **public static void** sendJson( RabbitTemplate rabbitTemplate) **throws** Exception{    MessageProperties messageProperties = **new** MessageProperties();  messageProperties.setContentType(**"text/plain"**);  Message message = **new** Message(**"hello"**.getBytes(),messageProperties);  rabbitTemplate.send(**""**,**"zhihao.miao.order"**,message);  }     **public static void** main(String[] args) **throws** Exception{  AnnotationConfigApplicationContext context = **new** AnnotationConfigApplicationContext(com.zhihao.miao.day03.Application.**class**);   RabbitTemplate rabbitTemplate = context.getBean(RabbitTemplate.**class**);  System.***out***.println(rabbitTemplate);   *//sendOrder(rabbitTemplate);  //sendUser(rabbitTemplate);  //sendOrderList(rabbitTemplate);  //sendOrderMap(rabbitTemplate);  sendJepg*(rabbitTemplate);   context.close();  } } |

消费端：

|  |
| --- |
| @Configuration **public class** MQConfig {   @Bean  **public** ConnectionFactory connectionFactory(){  CachingConnectionFactory factory = **new** CachingConnectionFactory();  factory.setUri(**"amqp://zhihao.miao:123456@192.168.1.131:5672"**);  **return** factory;  }   @Bean  **public** RabbitAdmin rabbitAdmin(ConnectionFactory connectionFactory){  RabbitAdmin rabbitAdmin = **new** RabbitAdmin(connectionFactory);  **return** rabbitAdmin;  }   @Bean  **public** RabbitTemplate rabbitTemplate(ConnectionFactory connectionFactory){  RabbitTemplate rabbitTemplate = **new** RabbitTemplate(connectionFactory);  **return** rabbitTemplate;  }   @Bean  **public** SimpleMessageListenerContainer messageListenerContainer(ConnectionFactory connectionFactory){  SimpleMessageListenerContainer container = **new** SimpleMessageListenerContainer();  container.setConnectionFactory(connectionFactory);  container.setQueueNames(**"zhihao.miao.order"**);   MessageListenerAdapter adapter = **new** MessageListenerAdapter(**new** MessageHandler());  *//指定Json转换器* Jackson2JsonMessageConverter jackson2JsonMessageConverter =**new** Jackson2JsonMessageConverter();    Map<String, Class<?>> idClassMapping = **new** HashMap<>();  idClassMapping.put(**"order"**,Order.**class**);  idClassMapping.put(**"user"**,User.**class**);   DefaultJackson2JavaTypeMapper jackson2JavaTypeMapper = **new** DefaultJackson2JavaTypeMapper();  jackson2JavaTypeMapper.setIdClassMapping(idClassMapping);   jackson2JsonMessageConverter.setJavaTypeMapper(jackson2JavaTypeMapper);  adapter.setMessageConverter(jackson2JsonMessageConverter);   TextMessageConverter textMessageConverter = **new** TextMessageConverter();   ContentTypeDelegatingMessageConverter contentTypeDelegatingMessageConverter = **new** ContentTypeDelegatingMessageConverter();  contentTypeDelegatingMessageConverter.addDelegate(**"text"**,textMessageConverter);  contentTypeDelegatingMessageConverter.addDelegate(**"html/text"**,textMessageConverter);  contentTypeDelegatingMessageConverter.addDelegate(**"xml/text"**,textMessageConverter);  contentTypeDelegatingMessageConverter.addDelegate(**"text/plain"**,textMessageConverter);   contentTypeDelegatingMessageConverter.addDelegate(**"json"**,jackson2JsonMessageConverter);  contentTypeDelegatingMessageConverter.addDelegate(**"application/json"**,jackson2JsonMessageConverter);   contentTypeDelegatingMessageConverter.addDelegate(**"image/jpg"**,**new** JPGMessageConverter());  contentTypeDelegatingMessageConverter.addDelegate(**"image/jepg"**,**new** JPGMessageConverter());  contentTypeDelegatingMessageConverter.addDelegate(**"image/png"**,**new** JPGMessageConverter());    adapter.setMessageConverter(contentTypeDelegatingMessageConverter);  *//设置处理器的消费消息的默认方法* adapter.setDefaultListenerMethod(**"onMessage"**);  container.setMessageListener(adapter);   **return** container;  } } |

指定的TextMessageConverter消息转换器

|  |
| --- |
| **public class** TextMessageConverter **implements** MessageConverter {    @Override  **public** Message toMessage(Object object, MessageProperties messageProperties) **throws** MessageConversionException {  System.***out***.println(**"=======toMessage========="**);  **return new** Message(object.toString().getBytes(),messageProperties);  }   @Override  **public** Object fromMessage(Message message) **throws** MessageConversionException {  System.***out***.println(**"=======fromMessage========="**);  **return new** String(message.getBody());  } } |

指定的JPGMessageConverter消息转换器

|  |
| --- |
| **public class** JPGMessageConverter **implements** MessageConverter{  @Override  **public** Message toMessage(Object object, MessageProperties messageProperties) **throws** MessageConversionException {  **return null**;  }   @Override  **public** Object fromMessage(Message message) **throws** MessageConversionException {  System.***out***.println(**"====JPGMessageConverter===="**);  **byte**[] body = message.getBody();  String fileName = UUID.randomUUID().toString();  String path = **"/Users/naeshihiroshi/Desktop/file/"**+fileName+**".jpg"**;  File file = **new** File(path);  **try**{  Files.copy(**new** ByteArrayInputStream(body),file.toPath());  }**catch** (IOException e){  e.printStackTrace();  }  **return** file;  } } |

客户端消息处理器

|  |
| --- |
| **public class** MessageHandler {    **public void** onMessage(**byte**[] message){  System.***out***.println(**"---------onMessage----byte-------------"**);  System.***out***.println(**new** String(message));  }    **public void** onMessage(String message){  System.***out***.println(**"---------onMessage---String-------------"**);  System.***out***.println(message);  }   **public void** onMessage(Order order){  System.***out***.println(**"---------onMessage---Order-------------"**);  System.***out***.println(order);  }   **public void** onMessage(User user){  System.***out***.println(**"---------onMessage---user-------------"**);  System.***out***.println(user);  }   **public void** onMessage(List<Order> orders){  System.***out***.println(**"---------onMessage---List<Order>-------------"**);  orders.stream().forEach(order -> System.***out***.println(order));  }   **public void** onMessage(Map<String,Object> orderMaps){  System.***out***.println(**"-------onMessage---Map<String,Object>------------"**);  orderMaps.keySet().forEach(key -> System.***out***.println(orderMaps.get(key)));  }   **public void** onMessage(File message){  System.***out***.println(**"-------onMessage---File message------------"**);  System.***out***.println(message.getName());  } } |

服务器端应用类，

|  |
| --- |
| @ComponentScan **public class** Application {  **public static void** main(String[] args) **throws** Exception{  AnnotationConfigApplicationContext context = **new** AnnotationConfigApplicationContext(Application.**class**);   System.***out***.println(**"===start up ing======"**);  TimeUnit.SECONDS.sleep(60);  context.close();  } } |

结论：

1、ContentTypeDelegatingMessageConverter是一个代理的MessageConverter。

2、ContentTypeDelegatingMessageConverter本身不做消息转换的具体动作，而是将消息转换委托给具体的MessageConverter。我们可以设置COntentType和MessageConverter的映射关系。

3、ContentTypeDelegatingMessageConverter还有一个默认的MessageConverter，也就是说当根据ContentType没有找到映射的MessageConverter的时候，就会使用默认的MessageConverter。