spring boot 整合rabbitmq

spring boot的环境怎么搭建这边就不提了， 这里引入spring boot -AMQP的依赖：

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

配置连接,创建交换机，队列

首先 我和上面一样 先要配置连接信息这里 可以选用yml的方式 也可以选用javaConfig的方式 这里两种方式我都贴出来 你们自己选

yml： 参数什么意思刚刚介绍过了 这里吧你自己的参数填进去就好了

```
spring:
  rabbitmq:
    host:
    port:
    username:
    password:
    virtual-host:
```

这样 spring boot 会帮你把rabbitmq其他相关的东西都自动装备好，

javaConfig：

```
@Bean
public ConnectionFactory connectionFactory() {
    CachingConnectionFactory connectionFactory = new
CachingConnectionFactory("localhost",5672);
    //我这里直接在构造方法传入了
    //        connectionFactory.setHost();
    //        connectionFactory.setPort();
    connectionFactory.setUsername("admin");
    connectionFactory.setPassword("admin");
    connectionFactory.setVirtualHost("testhost");
    //是否开启消息确认机制
    //connectionFactory.setPublisherConfirms(true);
    return connectionFactory;
}
```

配置完连接之后 我们就可以开始发送消息和接收消息了（因为我们上面刚刚测试rabbitmq的时候创建过队列和交换机等等这种东西了 当然 spring boot也可以创建）

spring boot创建交换机 队列 并绑定：

```
@Bean
public DirectExchange  defaultExchange() {
    return new DirectExchange("directExchange");
}

@Bean
public Queue queue() {
    //名字   是否持久化
    return new Queue("testQueue", true);
}

@Bean
public Binding binding() {
    //绑定一个队列   to: 绑定到哪个交换机上面 with: 绑定的路由建（routingKey）
    return BindingBuilder.bind(queue()).to(defaultExchange()).with("direct.key");
}
```

发送消息:

发送消息比较简单， spring 提供了一个RabbitTemplate 来帮助我们完成发送消息的操作

如果想对于RabbitTemplate 这个类进行一些配置（至于有哪些配置我们后面会讲到） 我们可以在config类中 吧他作为Bean new出来并配置

```
@Bean
public RabbitTemplate rabbitTemplate(ConnectionFactory connectionFactory) {
    //注意   这个ConnectionFactory 是使用javaconfig方式配置连接的时候才需要传入的   如果是yml配置的连接的话是不需要的
    RabbitTemplate template = new RabbitTemplate(connectionFactory);
    return template;
}
```

```
@Component
public class TestSend {

    @Autowired
    RabbitTemplate rabbitTemplate;

    public void testSend() {
        //至于为什么调用这个API 后面会解释
        //参数介绍： 交换机名字，路由建， 消息内容
        rabbitTemplate.convertAndSend("directExchange", "direct.key", "hello");
    }
}
```

我们只需要写一个类 然后交给spring 管理 在类里面注入RabbitTemplate 就可以直接调用api来发送消息了

接受消息：

这里我新建了一个项目（最好新建一个 当然 不新建也没关系） 来接收信息（之前的配置这里就不贴出来了 和上面基本一样），

```
@Component
public class TestListener  {


 @RabbitListener(queues = "testQueue")
    public void get(String message) throws Exception{
        System.out.println(message);
    }

}
```

不出意外的话运行起来能看见效果， 这里就不贴效果图了

那么至此rabbitmq的一个快速开始 以及和spring boot整合 就完毕了， 下面开始会讲一些rabbitmq的一些高级特性 以及原理等

RabbitMq特性

如何确保消息一定发送到Rabbitmq了?

我们刚刚所讲过的例子 在正常情况下 是没问题的， 但是 实际开发中 我们往往要考虑一些非正常的情况， 我们从消息的发送开始：

默认情况下，我们不知道我们的消息到底有没有发送到rabbitmq当中， 这肯定是不可取的， 假设我们是一个电商项目的话 用户下了订单 订单发送消息给库存 结果这个消息没发送到rabbitmq当中 但是订单还是下了，这时候 因为没有消息 库存不会去减少库存， 这种问题是非常严重的， 所以 接下来就讲一种解决方案： 失败回调

失败回调， 顾名思义 就是消息发送失败的时候会调用我们事先准备好的回调函数，并且把失败的消息 和失败原因等 返回过来。

具体操作：

注意 使用失败回调也需要开启发送方确认模式 开启方式在下文

更改RabbitmqTemplate:

```
@Bean
public RabbitTemplate rabbitTemplate(ConnectionFactory connectionFactory) {
    RabbitTemplate template = new RabbitTemplate(connectionFactory);
    //开启mandatory模式（开启失败回调）
    template.setMandatory(true);
    //指定失败回调接口的实现类
    template.setReturnCallback(new MyReturnCallback());
    return template;
}
```

回调接口的实现类：

实现RabbitTemplate.ReturnCallback里面的returnedMessage方法即可 他会吧相关的参数都传给你

```
public class MyReturnCallback implements RabbitTemplate.ReturnCallback {

    @Override
    public void returnedMessage(Message message, int replyCode, String replyText,
String exchange, String routingKey) {
        System.out.println(message);
        System.out.println(replyCode);
        System.out.println(replyText);
        System.out.println(exchange);
        System.out.println(routingKey);
    }
}
```

这里模拟一个失败的发送： 当指定的交换机不能吧消息路由到队列时（没有指定路由建或者指定的路右键没有绑定对应的队列 或者压根就没有绑定队列都会失败） 消息就会发送失败 效果:

```
312
NO_ROUTE
directExchange
direct.key123123
```

分别打印的是错误状态码，错误原因（这里的原因是不能路由） 交换机名字 和路由建 （还有个参数是你发送出去的消息 因为太长了就没截图了.）

可能有些同学想到了一个答案------事物

没错事物的确能解决这个问题， 而且 恰巧rabbitmq刚好也支持事物， **但是！ 事物非常影响rabbitmq的性能** 有多严重？ 据我所查到的资料 （当然 只是我所了解的 同学们也可以自己去尝试测试结果） 开启rabbitmq事物的话 **对性能的影响超过100倍之多** 也就是说 开启事物后处理一条消息的时间 不开事物能处理100条（姑且这样认为吧）， 那么 这样是非常不合理的， 因为消息中间件的性能其实非常关键的（参考双11） 如果这样子做的话 虽然能确保消息100%投递成功 但是代价太大了！

那么除了事物还有什么解决方案吗？

rabbitmq其实还提供了一种解决方案， 叫:**发送方确认模式** 这种方式 对性能的影响非常小 而且也能确定消息是否发送成功

而且 发送方确认模式一般也会和失败回调一起使用 这样 就能确保消息100%投递了

发送方确认开启:

其实代码在上面配置连接的时候已经放出来了 就是在连接工厂那被注释的一行代码：

```
        connectionFactory.setPublisherConfirms(true);
```

如果是yml配置的话：

```
spring:
  rabbitmq:
    publisher-confirms: true
```

和失败回调一样 实现一个接口:

```
public class MyConfirmCallback implements RabbitTemplate.ConfirmCallback{

    @Override
    public void confirm(CorrelationData correlationData, boolean ack, String cause) {
        System.out.println(correlationData);
        System.out.println(ack);
        System.out.println(cause);
    }
}
```

在RabbitmqTemplate 设置一下

```
template.setConfirmCallback(new MyConfirmCallback());
```

而且我们可以在发送消息的时候附带一个CorrelationData参数 这个对象可以设置一个id，可以是你的业务id 方便进行对应的操作

```
CorrelationData correlationData = new CorrelationData(UUID.randomUUID().toString());
rabbitTemplate.convertAndSend("directExchange", "direct.key123123",
"hello",correlationData);
```

效果:

```
CorrelationData [id=8ff37382-5202-4245-a5e8-ed6eb08a3328]
true
null
```

这里会吧我们传入的那个业务id 以及ack（是否发送成功） 以及原因 返回回来

但是 要注意的是 confirm模式的发送成功 的意思是发送到RabbitMq（Broker）成功 而不是发送到队列成功

所以才有了上面我所说的那句 要和失败回调结合使用 这样才能确认消息投递成功了

可能这里有点绕， 简单的总结一下就是 confirm机制是确认我们的消息是否投递到了 RabbitMq（Broker）上面 而mandatory是在我们的消息进入队列失败时候不会被遗弃（让我们自己进行处理）

那么上面 就是rabbitmq在发送消息时我们可以做的一些处理， 接下来我们会讲到rabbitmq在接收（消费）消息时的一些特性